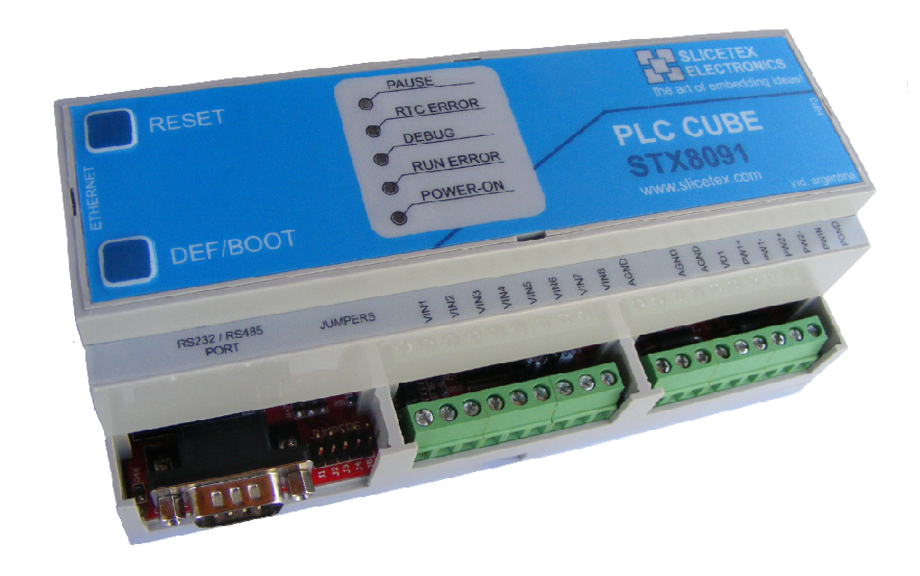
STX80XX
Power I/O Board & PLC Cube

Manual de Usuario Modo DAQ

Autor: Ing. Boris Estudiez



1 Descripción General

*El presente documento explica cómo programar los dispositivos de la línea **STX80XX** en Modo **DAQ**. En este modo de funcionamiento, es posible controlar el dispositivo para realizar funciones de control o adquisición de datos mediante una computadora conectada a una red Ethernet.*

Para enviar comandos al dispositivo, el usuario utiliza una librería de control que da acceso a todas las características del dispositivo de forma simple y fácil. A través de la librería, el usuario escribe sus aplicaciones o programas que permitan realizar el control deseado.

*Para utilizar la librería, se eligió un lenguaje de programación fácil, moderno y potente, llamado C# (C Sharp), disponible gratuitamente para Windows, a través del paquete de software **Microsoft Visual C# Express.** Es posible utilizar la librería con otros leguajes .Net como Microsoft Visual Basic.*

*Este documento le explicará cómo controlar la **STX80XX** desde sus programas escritos en C#.*

© 2015 Slicetex Electronics

2 Lecturas Recomendadas

Antes de leer este documento, recomendamos que se familiarice con el dispositivo de la línea **STX80XX** y el paquete de software SDK (**S**oftware **D**evelopment **K**it). Para ello recomendamos leer los siguientes documentos, en el orden detallado a continuación:

1. **STX80XX-GS-AX_BX_CX_DX**: Guía de Primeros Pasos para el modelo de dispositivo adquirido.
2. **STX80XX-DS-AX_BX_CX_DX**: Hoja de Datos para el modelo de dispositivo adquirido.

Para programar en Microsoft Visual C#, es necesario tener conocimientos básicos sobre el lenguaje de programación C# y tener instalado el entorno de desarrollo integrado (IDE).

Para facilitar su aprendizaje, hemos elaborado la siguiente guía básica, que recomendamos leer, si se programa por primera vez en C#:

1. **STX80XX-GS-CSHARP**: Guía Básica de C#.

Adicionalmente, puede adquirir libros de C# más avanzados de una librería o Internet. Muchas veces, las dudas sobre programación, ya están solucionadas y respondidas en Internet, utilizar el buscador Google (www.google.com) puede servirle de gran ayuda.

Mayor documentación puede encontrar en la pagina del producto, www.slicetex.com.

3 Requerimientos

Para programar en C#, es necesario tener instalado, el siguiente software:

1. **STX80XX-SDK** : Software Development Kit.
2. Microsoft Visual C# Express: Entorno de programación, versión gratuita disponible en internet.

Su instalación, es descripta en los documentos recomendados en la sección “Lecturas Recomendadas”.

Este manual presupone que:

- La versión de Firmware del dispositivo es la última disponible en el sitio Web.
- La versión de la librería STX8XXX.DLL es igual o superior a: 100.

4 Modo DAQ

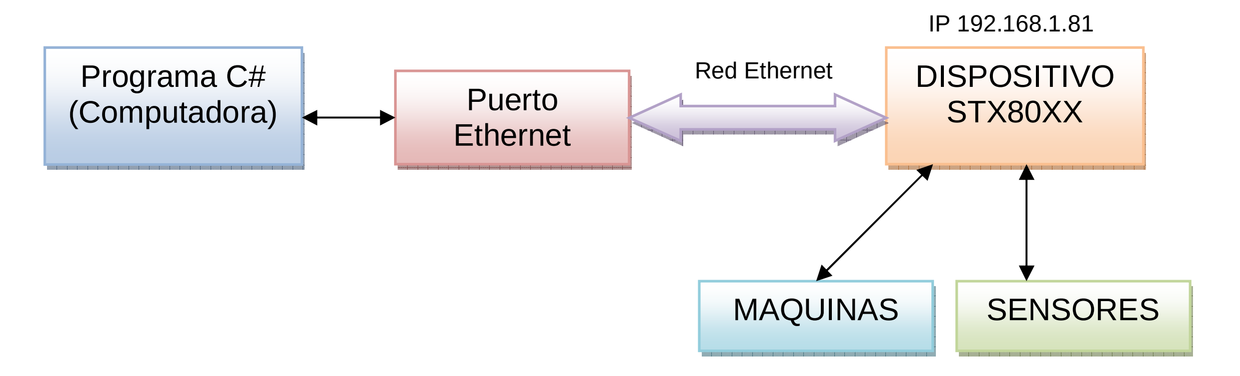
En esta sección se describirá la arquitectura del modo de funcionamiento DAQ de la línea STX80XX.

4.1 Definición

El modo de funcionamiento DAQ, es un modo de operación que se selecciona desde el programa ***BoardConfig*** o colocando un jumper (consulte hoja de datos del dispositivo). Se activará en la próxima inicialización del dispositivo (luego de un reset o power-up). *Este modo de funcionamiento solo está disponible en los modelos **AX** y **BX**.*

En el **Modo DAQ**, el dispositivo funciona como un sistema de adquisición de datos (DAQ), que le permitirá a través de una computadora conectada a una red Ethernet, realizar las siguientes tareas a través de comandos enviados al dispositivo:

- Controlar todas las características: Salidas digitales, Salidas RELAYS, Salidas PWM, Salidas Analógicas, Leer entradas analógicas y digitales, operar el generador sinusoidal, etc.
- Adquirir datos de señales y almacenarlos en una computadora.
- Realizar aplicaciones de control.
- Enviar datos de sensores a través de Internet, etc.



En el anterior diagrama en bloques, se observa que el usuario, realiza su programa en C#, lo ejecuta en una computadora y luego comanda a través de la interfaz Ethernet al dispositivo STX80XX. Por ejemplo puede controlar Maquinas y leer valores de Sensores varios.

Importante: En el modo DAQ, es necesario el uso de una computadora para operar el dispositivo STX80XX.

Si no desea utilizar una computadora para controlar el dispositivo, puede cambiar el modo de funcionamiento a **Modo PLC**, el cual trabaja de forma autónoma, corriendo un programa realizado en el lenguaje Ladder o Pawn. Consulte los documentos **STXLADDER-UM** y **STX80XX-MP-PLC-AX_CX_DX**, para más información.

4.2 Arquitectura del Modo DAQ

La conexión al dispositivo se realiza a través de una interfaz Ethernet. Sobre la interfaz Ethernet, se encapsulan los datos con el protocolo IP (Internet Protocol) y el protocolo UDP (User Datagram Protocol). Ambos protocolos en conjunto permiten transportar los datos a través de la red, y se lo denomina stack UDP/IP.

Para poder recibir comandos y ejecutarlos, el dispositivo tiene asignada una dirección IP (por defecto 192.168.1.81) y un puerto UDP (4950) donde escucha la llegada comandos.

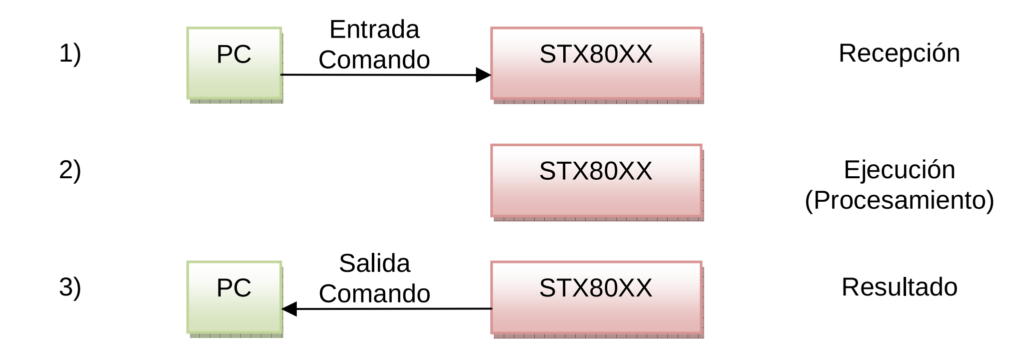
El protocolo UDP, permite solamente transportar datos y es sumamente eficiente en cuanto a velocidad, ya que es un protocolo “sin conexión”. Al no estar orientado a conexiones, los datos solamente se envían sin comprobar si llegaron o no al destino.

Para poder comprobar si los datos llegaron al destino, se diseño un protocolo especifico que funciona sobre el stack UDP/IP. El protocolo diseñado por Slicetex Electronics se llama **CSP** (**C**ommand **S**erver **P**rotocol).

El protocolo CSP, tiene las siguientes características principales:

- Permite comprobar si los datos llegaron a destino.
- Orientado a transmitir comandos y recibir sus resultados.
- Permite conocer el estado del comando enviado al dispositivo
- Permite establecer una clave (password) de 32-bit para asegurar que solo el usuario que conozca la clave pueda ejecutar los comandos enviados al dispositivo.

Cuando se envía un comando al dispositivo STX80XX utilizando el protocolo CSP, el siguiente mecanismo se produce:



Tres eventos ocurren:

1. Envío de comando desde PC al dispositivo STX80XX. Ej: Leer entrada analógica 1.
2. Ejecución del comando dentro del dispositivo STX80XX. Ej: El dispositivo lee la entrada analógica 1.
3. Respuesta del comando enviado desde el dispositivo STX80XX al PC. Ej: El dispositivo devuelve el valor de entrada analógica.

5 Primer Programa

La mejor forma de aprender es practicando, por lo tanto en esta sección mostraremos un procedimiento practico para crear el primer programa de prueba en C#.

El programa se comunicará con la el dispositivo STX80XX y enviara dos comandos. El primer comando leerá una entrada discreta, entrada DIN1. El segundo comando, cambiara el estado del RELAY1, activando o desactivando, según su último estado.

Adjunto a este documento, podrá encontrar el programa completo, listo para compilar y ejecutar en Microsoft Visual C# 2005. El proyecto se llama "**Prueba1**".

Asegúrese de entender bien el procedimiento mostrado, ya que el resto de los ejemplos descriptos en este documento, parten de la suposición que usted entiende el proceso para enviar comandos al dispositivo STX80XX mediante C#.

Importante: Este ejemplo muestra como conmutar el estado del RELAY1, pero quizás algunos dispositivos no contengan una salida relé, en dicho caso, puede utilizar una salida digital genérica del tipo DOUT1. El resultado es el mismo. Consulte la hoja de datos del dispositivo para más información.

5.1 Requerimientos Previos

Para poder enviar comandos al dispositivo es necesario tener instalado el paquete de software STX80XX-SDK (Software Development Kit) y Microsoft Visual C# Express.

Se recomienda leer los siguientes documentos antes de continuar:

1. **STX80XX-GS-AX_BX_CX** : Guía de Primeros Pasos.
2. **STX80XX-GS-CSHARP**: Guía Básica de C#.

5.2 Librería STX8XXX.DLL

La librería ST8XXX.DLL es un archivo DLL (Dynamic Link-Library), que contiene la interfaz necesaria para que usted pueda controlar dispositivos de la línea STX80XX desde sus programas. *Esta librería debe ser enlazada o referenciada, desde cada programa que usted cree.*

La librería contiene funciones, métodos, objetos y rutinas, que le permiten controlar fácilmente el dispositivo.

La librería **STX8XXX.DLL** se encuentra dentro del directorio donde usted instalo el SDK (Software Development Kit), en la carpeta:

visual_cs\libs\stx8xxx

5.3 Diseñar el Programa

Lo primero que debe hacerse al crear un programa con interfaz grafica, es diseñar la interfaz visual. Para ello abriremos entorno de desarrollo "Microsoft Visual C# Express", haciendo doble click en el icono del programa:

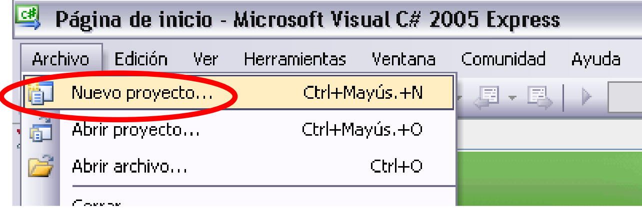


Importante: En este documento se utiliza "Microsoft Visual C# Express 2005", usted puede utilizar cualquier otra versión, el concepto es el mismo.

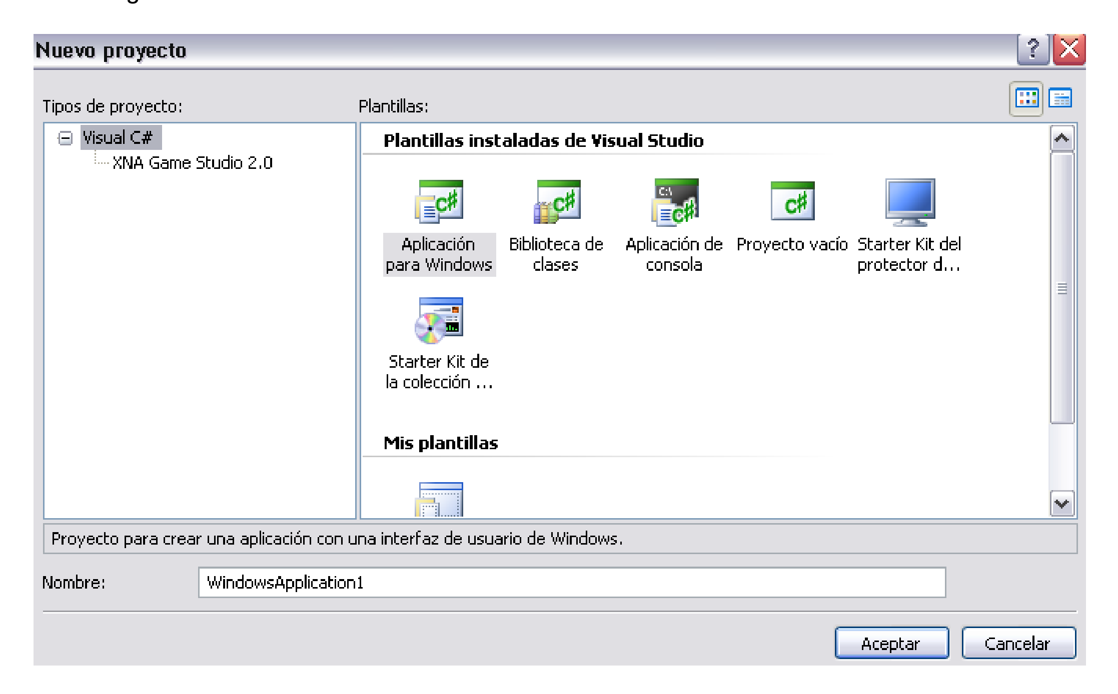
La siguiente pantalla aparecerá al abrir el entorno:



Para crear un nuevo programa, seleccionamos el menú "Archivo" y luego hacemos click en "Nuevo proyecto":

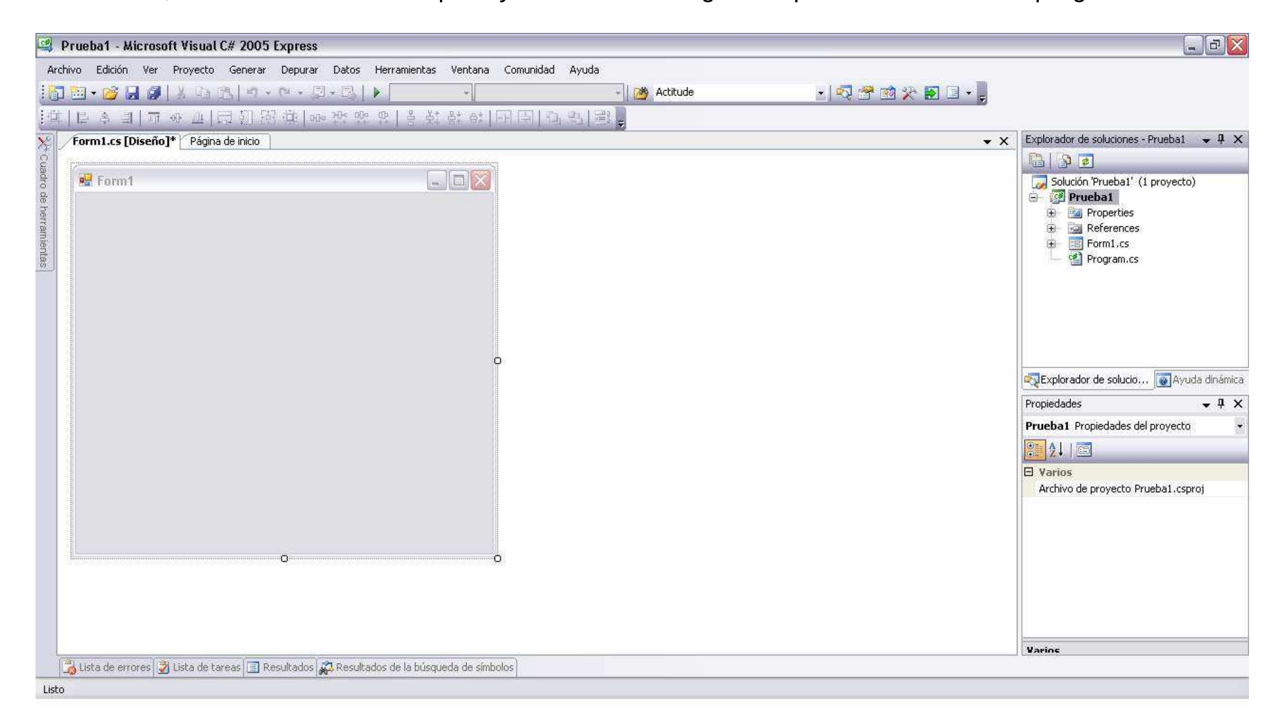


Aparecerá la siguiente ventana:



En tipos de proyecto elegimos "Visual C", en plantillas seleccionamos "Aplicación para Windows", en nombre escribimos "Prueba1", el mismo será utilizado como nombre de nuestro programa.

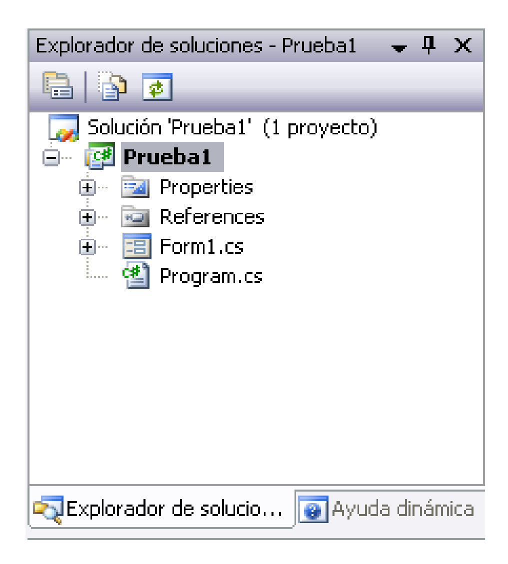
Finalmente, hacemos click en "Aceptar" y obtenemos la siguiente pantalla con nuestro programa:



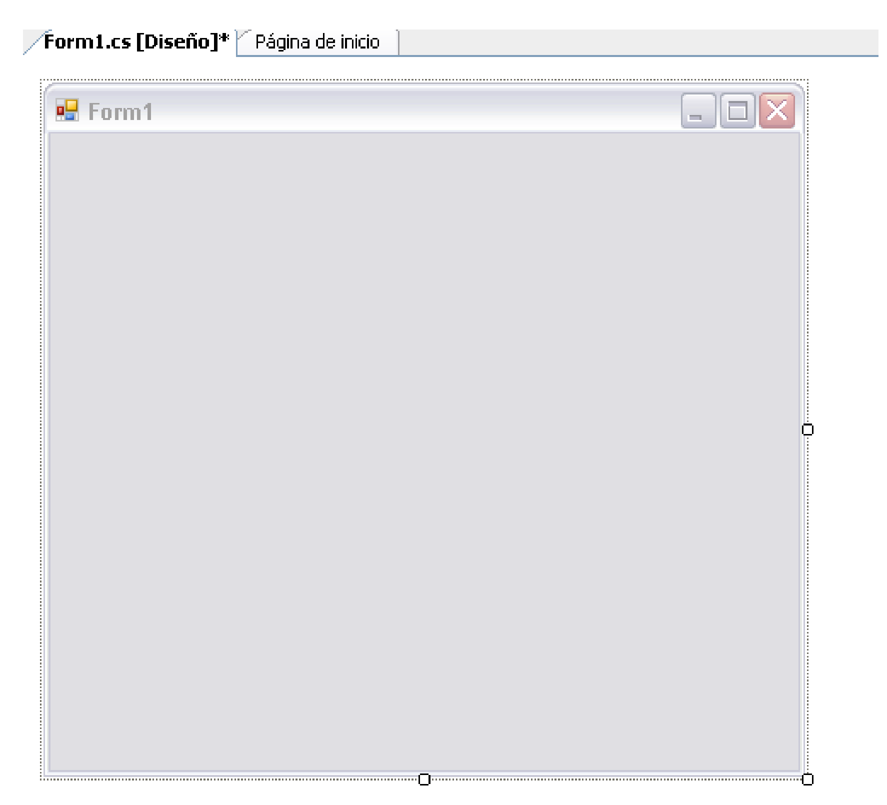
© 2015 Slicetex Electronics

En la pantalla anterior, hay cuatro secciones que debe identificar:

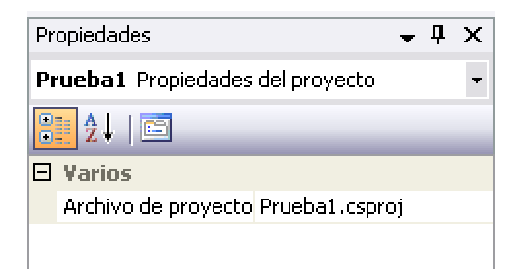
1 - Explorador de Soluciones: Aquí se listan todos los archivos del proyecto.



2 - Formulario: Aquí está la ventana principal de nuestro programa, donde usted podrá colocar botones, cajas de texto, etc.

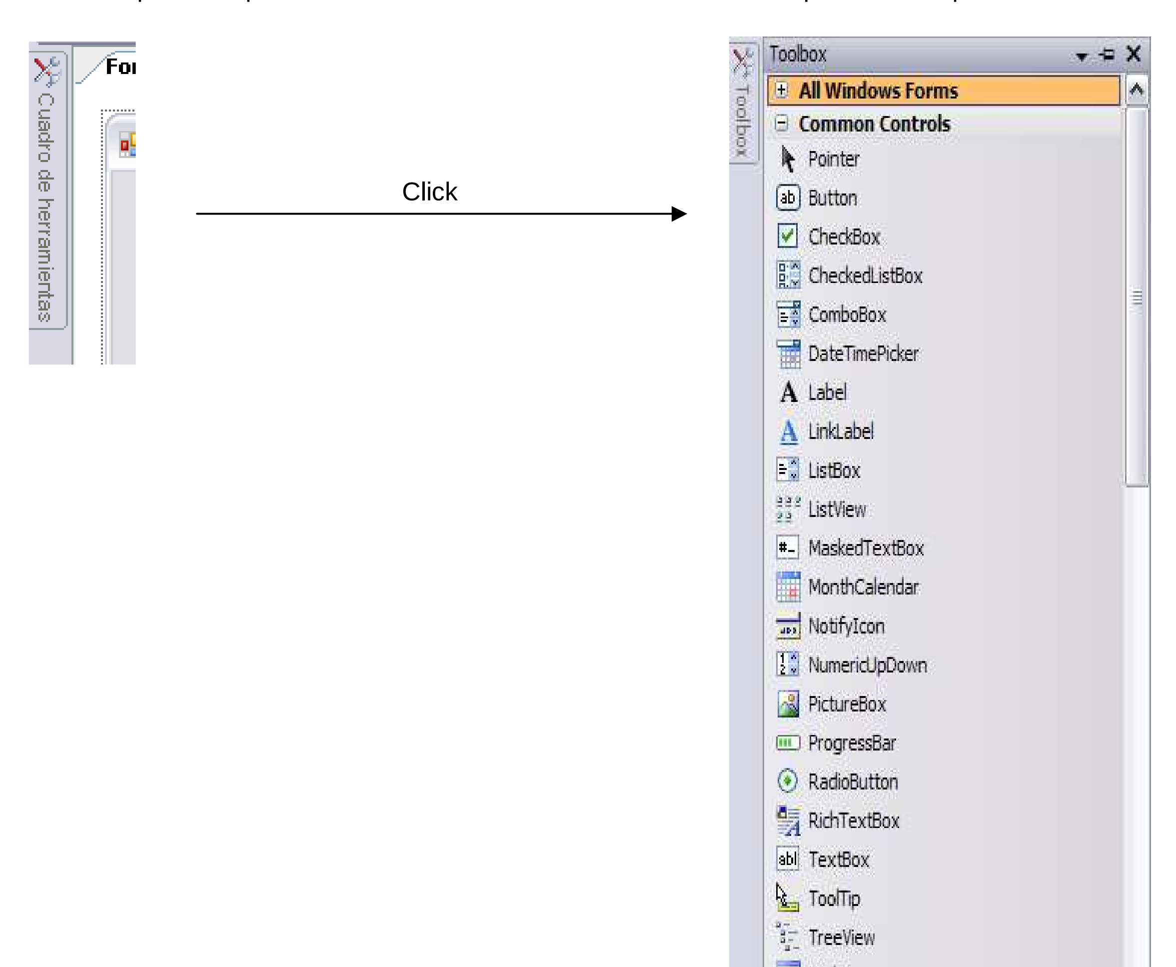


3 – Ventana de Propiedades: Aquí aparecerán las propiedades de cada objeto grafico que coloque en el formulario (ancho de ventana, tipo de letra de un cuadro de texto, etc.).



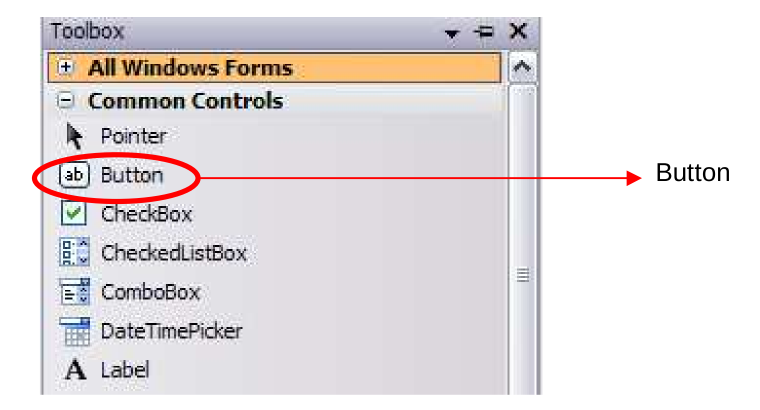
4 – Cuadro de Herramientas: Contiene los objetos que usted puede utilizar en su formulario.

Click en la pestaña que dice “Cuadro de Herramientas” al costado izquierdo de la pantalla:

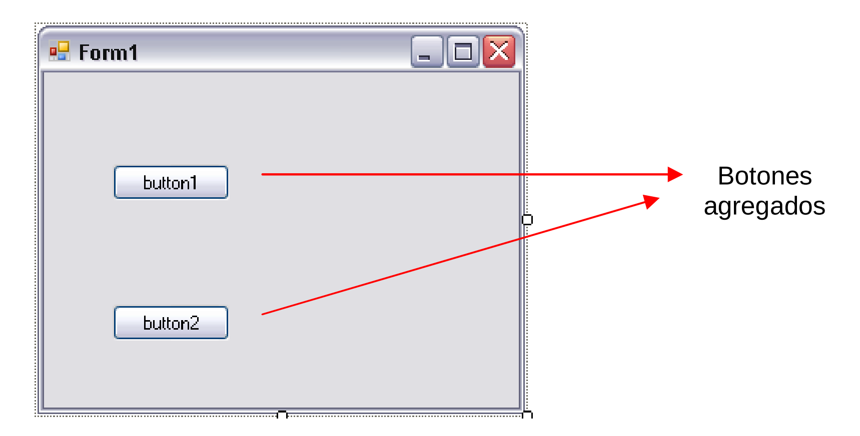


El próximo paso será colocar dos botones y una caja de texto en el formulario:

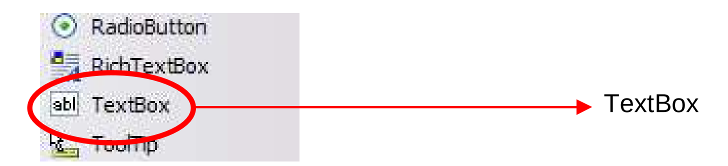
Para ello vamos al "Cuadro de Herramienta" y seleccionamos "Button", que es el botón:



Hacemos click en "Button" y mantenemos apretado el Mouse y lo arrastramos al formulario. Repetimos el procedimiento una vez más, para agregar dos botones, y nuestro formulario debería verse de la sig. forma:

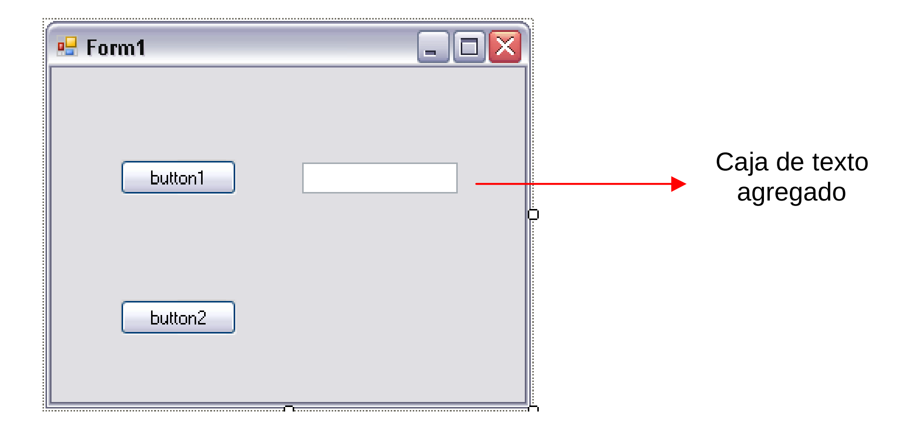


Ahora agregaremos una caja de texto, llamada "TextBox":



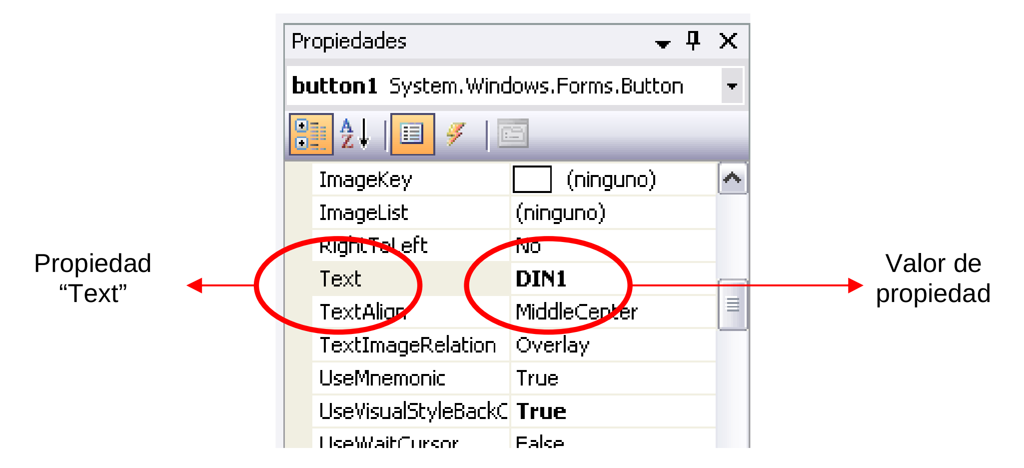
Al igual que con el botón, hacemos click en la "TextBox" y la arrastramos al formulario.

Finalmente, el formulario nos queda de la siguiente forma:

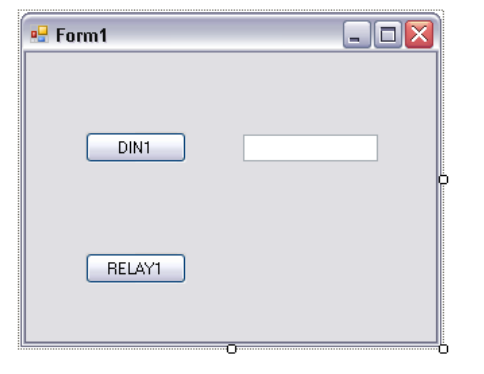


Ahora, cambiaremos los nombres mostrados de los objetos en el formulario, para mejor visualización:

Click en el “botton1” y luego desde la “Ventanas de Propiedades”, buscamos la propiedad “Text” y su valor asignado, lo cambiamos por “DIN1”:



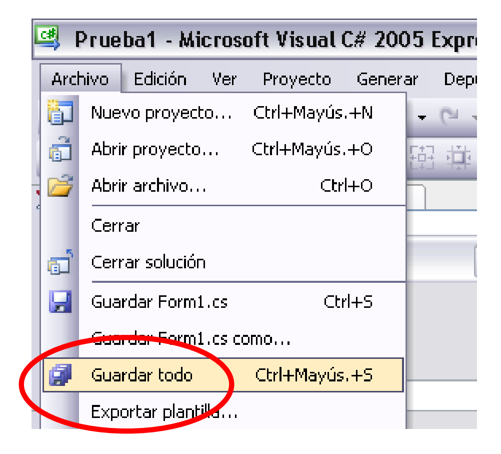
Procedemos de la misma manera para el “botton2”, pero esta vez, cambiamos el valor de “Text” a “RELAY1”. La pantalla del programa lucirá de la siguiente forma:



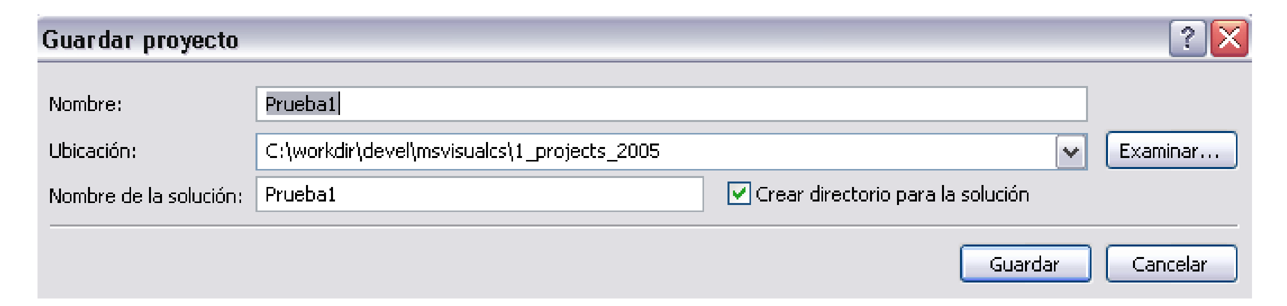
Nota: Cambiar el valor de la propiedad “Text” de un objeto, solo cambia el texto mostrado en pantalla. Para cambiar el nombre de un objeto, hay que cambiar la propiedad “Name”.

5.4 Guardar el Proyecto

Para guardar el proyecto, haga click en "Archivo" y luego en "Guardar todo":



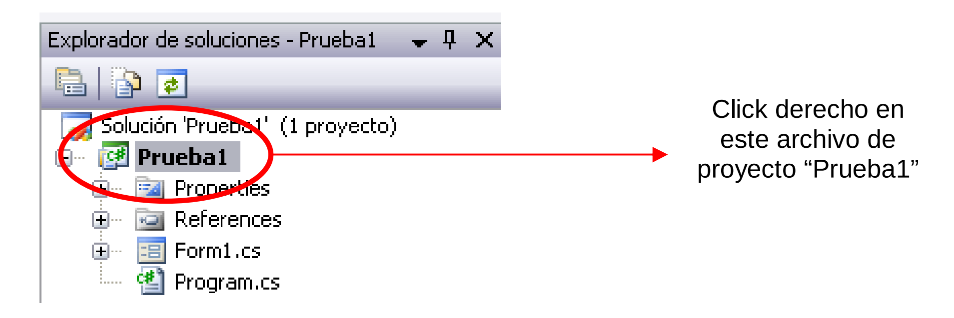
Le aparecerá la siguiente pantalla:



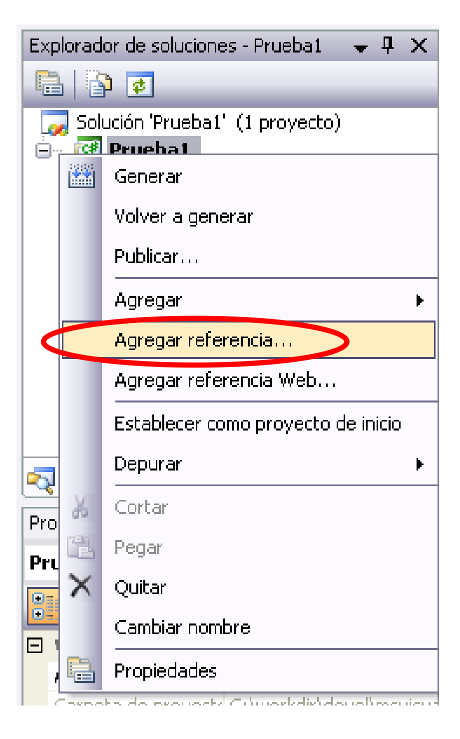
Elija la ubicación o directorio donde quiera guardar el proyecto (o solución en términos de Visual C#) y haga click en "Guardar". En este punto su proyecto fue guardado en disco.

5.5 Referenciar Librería STX8XXX.DLL

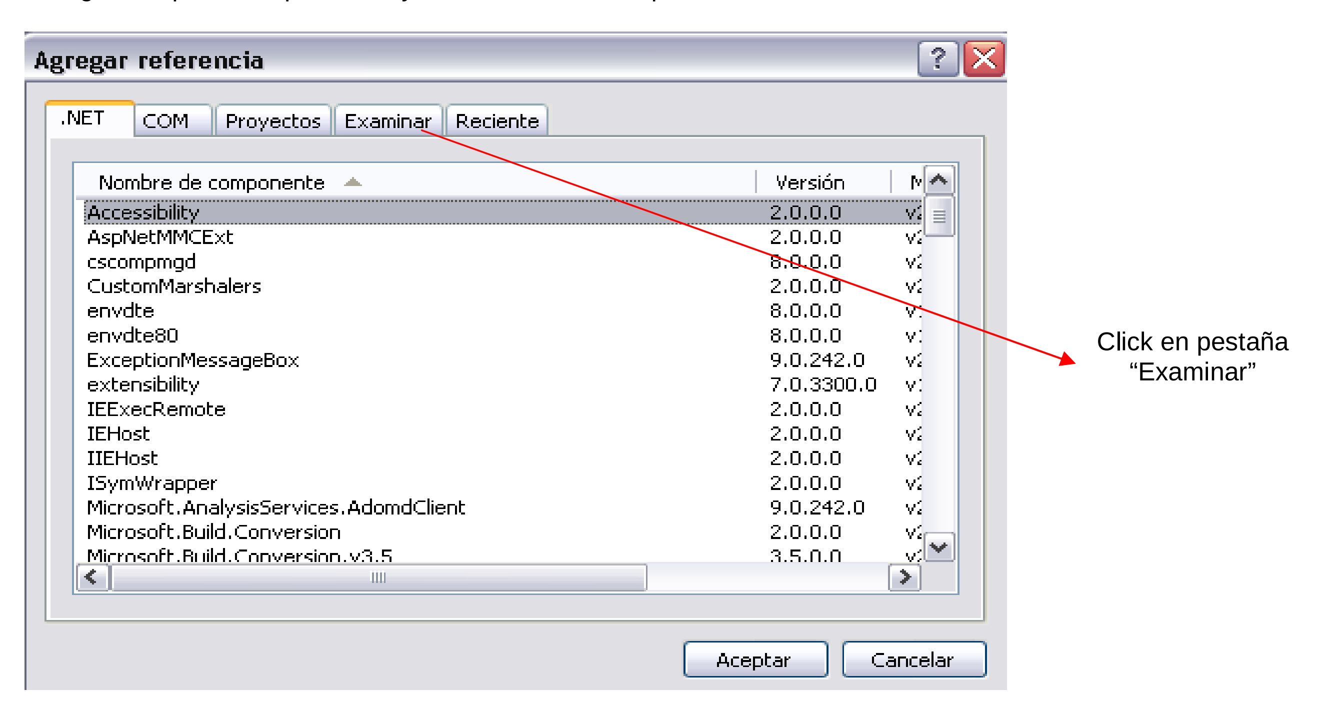
Ahora vamos a enlazar o referenciar el archivo STX8XXX.DLL que contiene la librería para operar el dispositivo STX80XX. Para ello nos dirigimos al explorador de soluciones "Explorador de Soluciones", y hacemos click derecho en "Prueba1":



Al hacer click derecho, el siguiente menú se desplegara, en donde haremos click en el ítem “Agregar referencia…”:



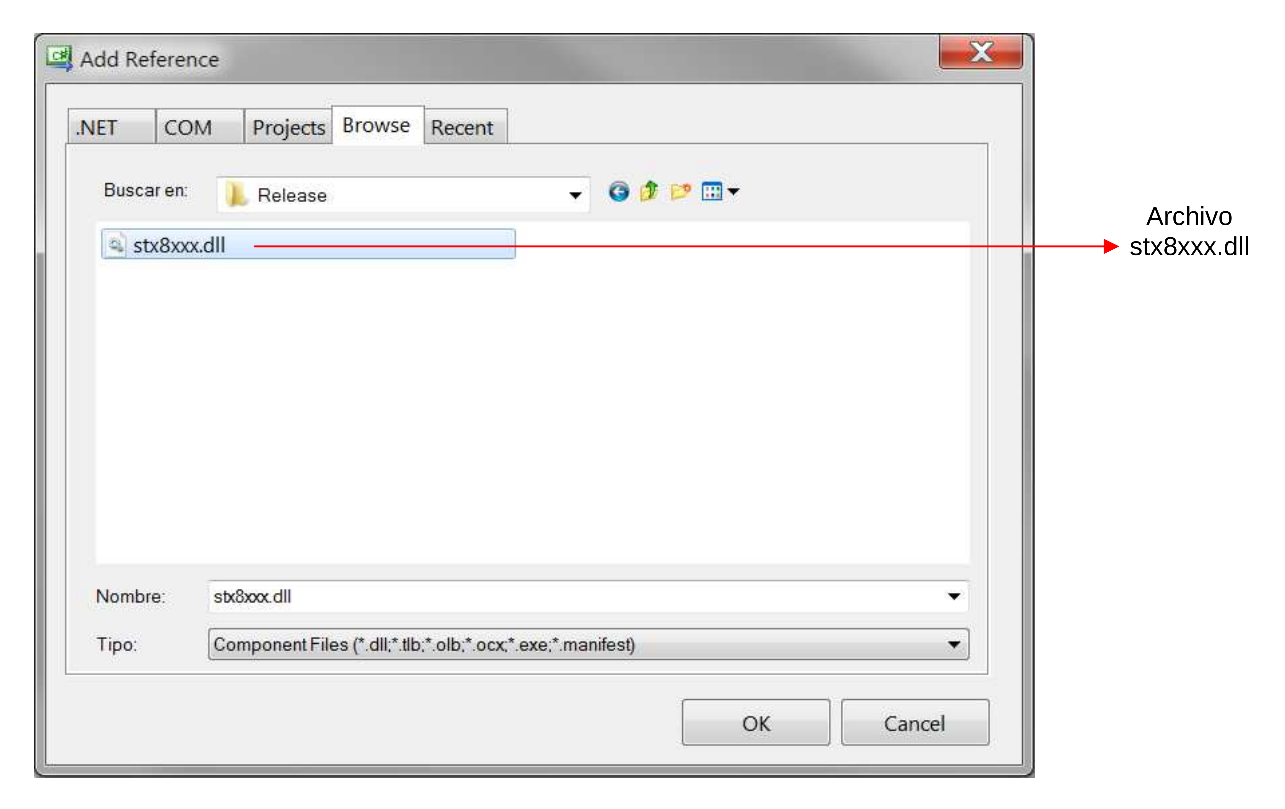
La siguiente pantalla aparecerá, y haremos click en la pestaña “Examinar”:



En la pestaña “Examinar” buscaremos el archivo “stx8xxx.dll”, que es la librería. Para ello nos deberemos deslazar al directorio donde se encuentra el software SDK (Software Development Kit) y bajo el directorio:

visual_cs\libs\stx8xxx

Seleccionaremos el archivo “stx8xxx.dll” como se muestra a continuación:

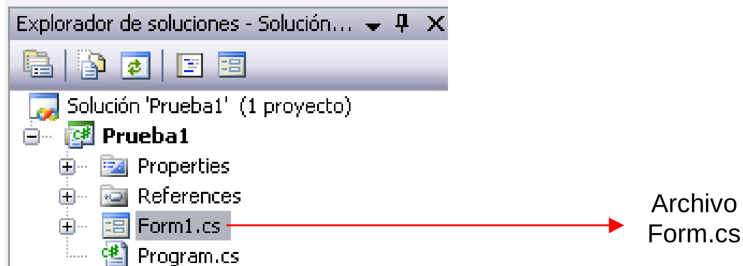


Luego click en “Aceptar” y listo, la librería fue agregada a nuestro proyecto.

5.6 Inicializar la Librería

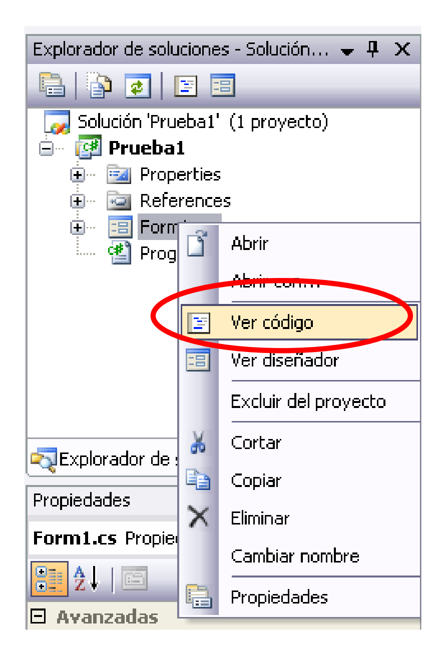
Ahora procederemos a agregar código en C#, para inicializar o poner a punto la librería.

Primero, vamos a agregarla al especio de nombres de nuestro programa, utilizando la directiva “using”. Para ello vamos al explorador de soluciones, y hacemos click en el archivo de nuestro formulario principal “Form1”:



El archivo Form.cs contiene el código C# de nuestra ventana o formulario principal del programa.

Hacemos click derecho y luego seleccionamos "Ver código":



Se debería abrir la siguiente ventana:

Form1.cs | Form1.cs [Diseño] | Página de inicio

Prueba1.Form1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Prueba1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

La misma muestra el código en C# que ejecuta nuestro programa actualmente. Para agregar nuestra librería al espacio de nombres del programa, utilizaremos la directiva "using", agregando "using stx8xxx;" al archivo Form.cs y el código nos quedaría como sigue:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using stx8xxx;
```

Directiva using agregada.

```
namespace Prueba1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

El próximo paso es crear un objeto llamado **PioBoard**, el cual contendrá todos los métodos y clases para acceder el dispositivo STX80XX. El nombre **PioBoard** significa “Power I/O Board”, pero usted puede nombrarlo con cualquier otro nombre.

Dentro de la clase “Form1”, creamos una variable global que será nuestro objeto del tipo **Stx8xxx**, y se llamara **PioBoard**.

```
Stx8xxx PioBoard;
```

El codigo resultante quedaria de la siguiente forma (se omiten las directivas “using” por simplicidad):

```
namespace Prueba1
{
    public partial class Form1 : Form
    {
        Stx8xxx PioBoard;

        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Ahora vamos a inicializar la librería, esto se hace en el constructor de la clase Form1 y con el siguiente codigo:

```
PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);
```

Donde asignamos a PioBoard un objeto proveniente de la clase Stx8xxx, al cual se le pasó la dirección IP del dispositivo “192.168.1.81” como primer argumento, como segundo argumento el password a utilizar para enviar comandos, en este caso 0. Finalmente como tercer y último argumento el tipo de dispositivo a controlar, en este caso un dispositivo STX8081. Pero puede elegir cualquier otro modelo.

Para facilitar la codificación, la librería STX8XXX.DLL tiene documentación en línea, la cual está disponible si ponemos el puntero del Mouse sobre algún nombre.

Por ejemplo, al poner el Mouse sobre Stx8081, aparece el siguiente cartel, que nos muestra que es un objeto “Power I/O Board”.

```
PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);
                Stx8xxx.Stx8xxx(string IP, uint Password, Stx8xxxId TargetDevice)
                STX8XXX Power I/O Board Object.  Also called PioBoard Object.
```

Si queremos información sobre sus argumentos, al escribir la línea, luego del “(“ nos aparecerá un cartel con las posibles formas de inicializar la librerías, apretamos la tecla “DOWN” y elegimos la opción “3”, apareciendo un cartel con la información del argumento:

```
PioBoard = new Stx8xxx(
           ▲3 of 5▼  Stx8xxx.Stx8xxx (string IP, uint Password, Stx8xxxId TargetDevice)
           IP: STX8XXX Board IP address.
```

En ingles, nos dice que el primer argumento es del tipo “string” llamado IP y significa “Dirección IP de del dispositivo STX80XXX”.

Finalmente, el código resultante es:

```
namespace Prueba1
{
    public partial class Form1 : Form
    {
        Stx8xxx PioBoard;

        public Form1()
        {
            InitializeComponent();

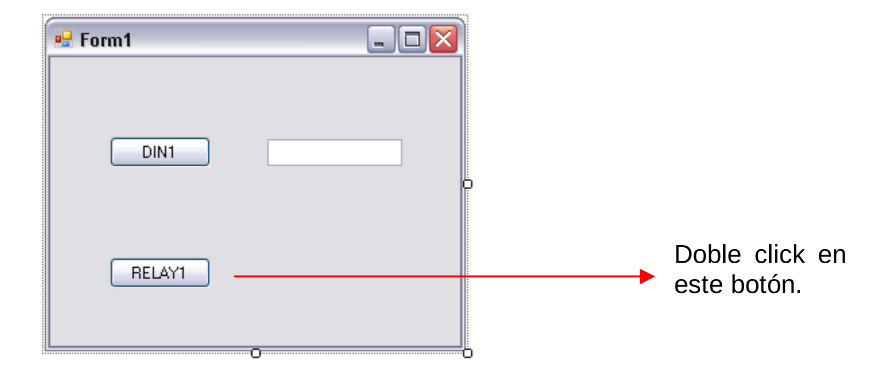
            PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);
        }
    }
}
```

En este punto, la librería fue creada e inicializada. Notar que la dirección IP, como el valor del password, son valores que vienen por defecto en el dispositivo, pero usted puede cambiarlos, desde el programa de configuración ***BoardConfig***.

5.7 Enviando Comandos

Nuestra próxima acción consistirá en enviar un comando al dispositivo, que permita cambiar el estado del RELAY1 cada vez que hagamos un click en el botón “RELAY1” de la ventana principal del programa.

Para lograrlo, deberemos asociar un evento “Click” al botón del formulario. Esto se puede hacer haciendo doble click sobre el botón “RELAY1” del formulario principal, en la vista de diseño:



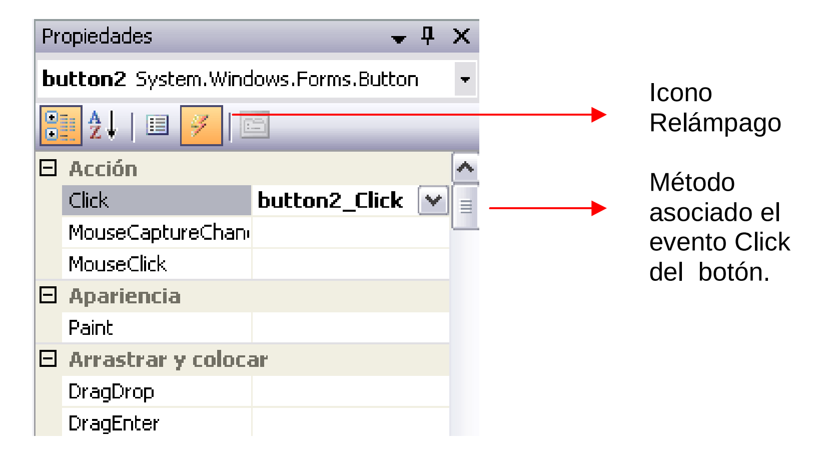
Automáticamente, se nos abrirá la ventana de código con el siguiente método creado, que representa al evento “Click” del botón:

```
private void button2_Click(object sender, EventArgs e)
{

}
```

Notar que el objeto botón “RELAY” se llama “button2”, por lo tanto el método creado automáticamente por el entorno Visual C#, comienza con el nombre del objeto: “button2”.

Para visualizar los eventos asociados a un objeto, ir a la ventana de propiedades y hacer click en el icono “relámpago”, donde se nos mostrara lo siguiente:



Ahora dentro del método del evento creado (`button2_Click`), debemos escribir el comando para conmutar el RELAY1 del dispositivo STX80XX, entonces escribimos la siguiente línea:

```
PioBoard.Cmd.Relay.Toggle(Relays.Relay1);
```

Resultando:

```
private void button2_Click(object sender, EventArgs e)
{
    PioBoard.Cmd.Relay.Toggle(Relays.Relay1);
}
```

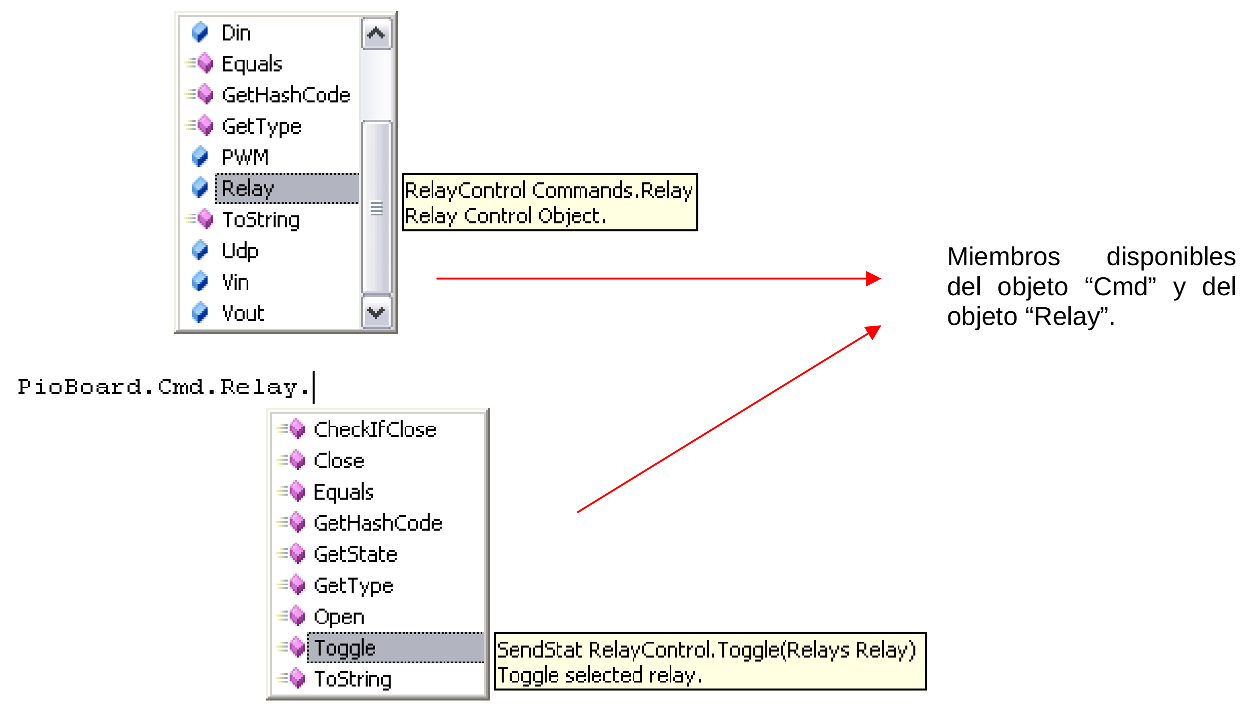
Eso es todo!.

El programa ya puede ser compilado, para conmutar el estado del RELAY1.

Pero antes, explicaremos que hicimos:

1. Llamamos al método “Toggle” que está dentro del objeto “Relay”. Al método le pasamos como primer argumento el nombre del relé a conmutar, es decir el Relay1.
2. El objeto “Relay” está dentro del objeto “Cmd” (commands), que a su vez está adentro del objeto “PioBoard” que fue creado con anterioridad.

Para acceder a objetos que están dentro de otros objetos, se utiliza el operador punto “.”. Cada vez que escribimos un “.” luego de un objeto, el entorno Visual C# nos muestra todos los miembros disponibles de ese objeto como se muestra a continuación:



5.8 Compilando el Programa

Para compilar el programa C# y luego ejecutarlo, hay dos alternativas, generar una versión "Debug", la cual es optima para depurar programas, o generar una versión "Release", que genera el archivo binario .EXE del programa, optimo para utilizarlo como una aplicación estándar de computadora.

El código completo del programa, es el siguiente:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using stx8xxx;

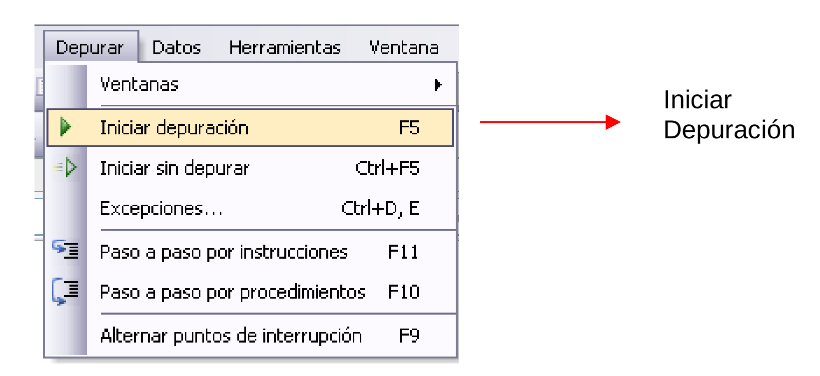
namespace Prueba1
{
    public partial class Form1 : Form
    {
        Stx8xxx PioBoard;

        public Form1()
        {
            InitializeComponent();

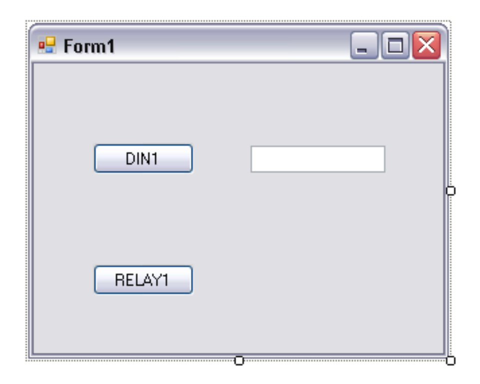
            PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            PioBoard.Cmd.Relay.Toggle(Relays.Relay1);
        }
    }
}
```

Para generar una versión "Debug", click en "Iniciar depuración", del menú "Depurar":



El programa se ejecutara, mostrando la ventana principal:

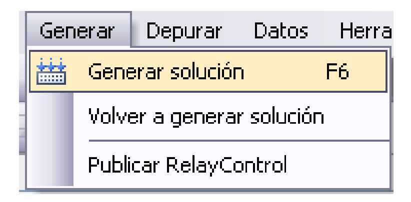


Al hacer click en el botón "RELAY1", se enviara el comando "Toggle" al dispositivo, el cual invierte el estado actual de la salida, es decir si está activado, lo desactiva, y si está desactivado, lo activa. Simple, ¿ no ?.

Si su dispositivo está conectado a la red Ethernet correctamente y a su computadora, podrá escuchar la conmutación del RELAY1 cada vez que haga un click en el botón. Adicionalmente, algunos dispositivos disponen de un Led asociado al RELAY1, que se activará o desactivará, dependiendo del estado del RELAY1.

Para detener o salir del programa, ciérrelo de la forma habitual, haciendo click en la X de la ventana.

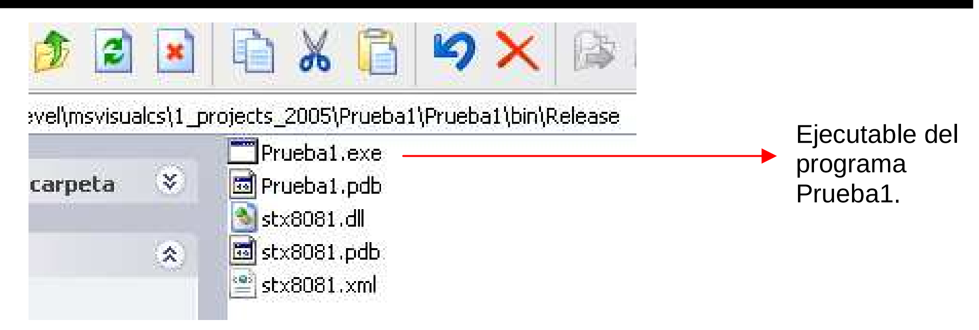
Para generar la versión "Release", click en "Generar solución" de menú "Generar":



El programa se compilara y creara el ejecutable **Prueba1.exe**. Busque el ejecutable en el directorio donde guardo la solución. Una vez dentro del directorio de la solución, bajo los directorios:

Prueba1\Prueba1\bin\Release

Se encontrara el ejecutable. Doble click en él y el programa se ejecutara nuevamente.


evel\msvisualcs\1_projects_2005\Prueba1\Prueba1\bin\Release
carpeta
Prueba1.exe
Prueba1.pdb
stx8081.dll
stx8081.pdb
stx8081.xml
Ejecutable del
programa
Prueba1.

5.9 Enviar Comandos y Recibir Resultados

El próximo paso en el desarrollo de nuestro programa consistiría en enviar un comando para leer el valor de la entrada discreta DIN1 y mostrar si su estado es bajo “0 lógico” o alto “1 lógico”.

El funcionamiento debería ser el siguiente, al hacer click sobre el botón “DIN1”, se enviara un comando para leer el estado de la entrada DIN1. Al retornar el comando, el estado de la entrada estará disponible en una variable. De acuerdo al valor de la variable, se imprimirá en la caja de texto (TextBox), el texto “1 lógico” o “0 lógico”.

Primero asociaremos el evento click al botón “DIN1”, el procedimiento es el mismo al empleado anteriormente con el botón “RELAY1”. Desde el formulario de diseño hacemos doble click en el botón “DIN1” y se abrirá la ventana de código, con el siguiente evento creado:

```
private void button1_Click(object sender, EventArgs e)
{

}
```

Para leer una entrada utilizamos el objeto “Din”, accesible desde nuestro objeto global “PioBoard”. El objeto “Din” tiene un método llamado “CheckIfHigh()”, que permite comprobar si una entrada discreta esta en alto o no.

El método en cuestión es el siguiente:

```
PioBoard.Cmd.Din.CheckIfHigh(DinInput.Din1, out Din1Stat);
```

Devuelve en la variable “Din1Stat” del tipo “bool”, pasada como segundo parámetro de retorno (**out**), el valor “true” si la entrada discreta esta en alto, o “false” si esta en bajo. La entrada discreta DIN1 es pasada como primer parámetro “DinInput.Din1”.

Entonces, colocamos el comando en el evento button1_click:

```
private void button1_Click(object sender, EventArgs e)
{
    bool Din1Stat;

    PioBoard.Cmd.Din.CheckIfHigh(DinInput.Din1, out Din1Stat);
}
```

Ahora debemos comprobar si DIN1 está en alto o bajo, y actualizar el “textBox1” para que muestre el resultado:

```
private void button1_Click(object sender, EventArgs e)
{
    bool Din1Stat;

    PioBoard.Cmd.Din.CheckIfHigh(DinInput.Din1, out Din1Stat);

    if (Din1Stat == true)
    {
        textBox1.Text = "1 logico";
    }
    else
    {
        textBox1.Text = "0 logico";
    }
}
```

Como muestra el código, comprobamos si el comando devuelve “true” con un condicional “if” sobre la variable “Din1Stat” del tipo “bool”.

En caso de retornar “true”, actualizamos el valor de la “textBox1” con el texto “1 logico”.

Caso contrario, actualizamos la “textBox1” con el texto “0 logico”.

La propiedad que cambia el valor de texto en una TextBox, se llama “Text”.

El codigo programa completo del programa es (no se incluyen las directivas using):

```
namespace Prueba1
{
    public partial class Form1 : Form
    {
        Stx8xxx PioBoard;

        public Form1()
        {
            InitializeComponent();

            PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);
        }

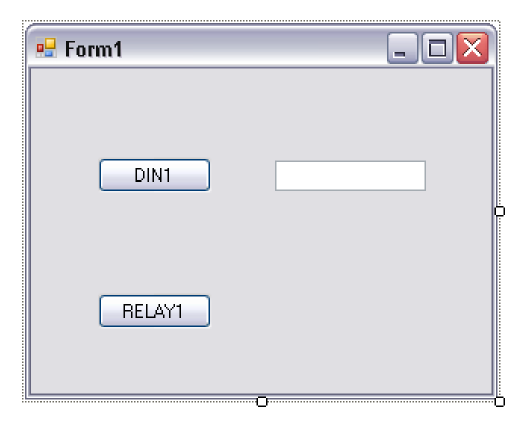
        private void button2_Click(object sender, EventArgs e)
        {
            PioBoard.Cmd.Relay.Toggle(Relays.Relay1);
        }

        private void button1_Click(object sender, EventArgs e)
        {
            bool Din1Stat;

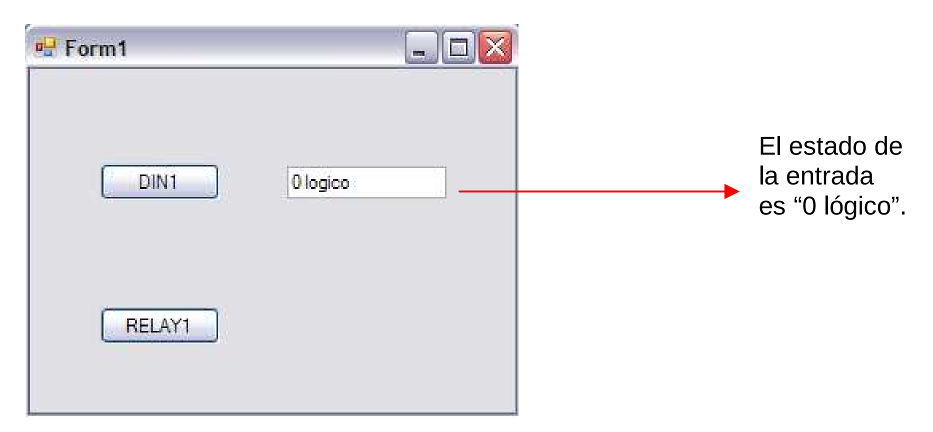
            PioBoard.Cmd.Din.CheckIfHigh(DinInput.Din1, out Din1Stat);

            if (Din1Stat == true)
            {
                textBox1.Text = "1 logico";
            }
            else
            {
                textBox1.Text = "0 logico";
            }
        }
    }
}
```

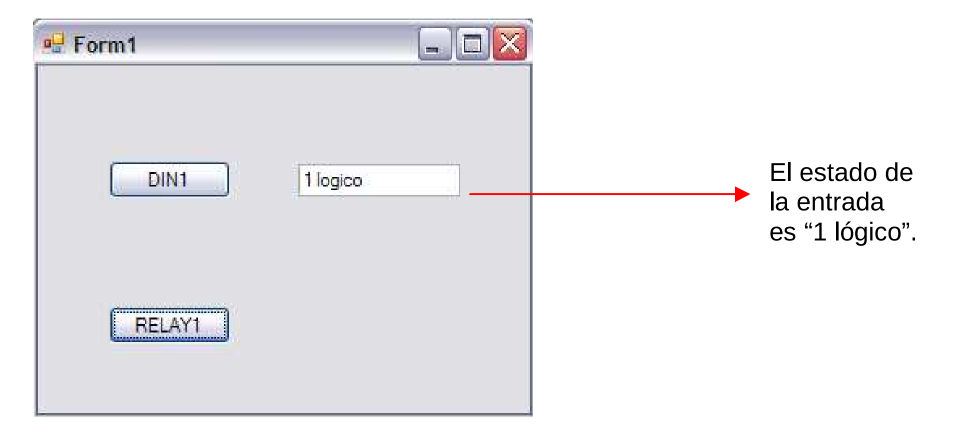
Luego compilamos el programa y lo ejecutamos, y si no aparece ningun error el programa deberia lucir como se muestra a continuacion:



Al hacer clic en el botón "DIN1", el programa debería mostrar el estado de la entrada. Si no tiene señal aplicada, debería mostrar:



Si la entrada esta polarizada, mostrará:



5.10 Comprobación de Errores

Hasta ahora hemos enviado los comandos sin comprobar si se ejecutaron correctamente o no. En aplicaciones profesionales, es necesario saber si el RELAY1 fue realmente conmutado o no. Para ello debemos comprobar el estado del comando cuando retorna e indicar si existió algún error.

Si la red no está disponible, o el dispositivo no está operable, o el comando fue rechazado, se retornará un código de error. Este error deberíamos procesarlo y mostrarlo al usuario, indicándole que algo salió mal.

Entonces, volviendo al comando para conmutar el RELAY1:

```
PioBoard.Cmd.Relay.Toggle(Relays.Relay1);
```

Si nos posicionamos sobre el método “Toggle” con el Mouse, nos aparece el siguiente cartelito:

```
PioBoard.Cmd.Relay.Toggle(Relays.Relay1);
SendStat RelayControl.Toggle(Relays Relay)
Toggle selected relay.
```

Donde podemos ver que el método “Toggle” retorna una variable del tipo “SendStat”. Una variable del tipo “SendStat” contiene el estado de una transacción al enviar un comando. Más adelante en este documento se explica con detenimiento la variable “SendStat”. Pero por ahora nos basta saber, que si esta variable, es distinta a “Success”, ocurrió un error al enviar el comando.

Entonces, escribimos el siguiente código:

```
private void button2_Click(object sender, EventArgs e)
{
     SendStat CmdStat;

     CmdStat = PioBoard.Cmd.Relay.Toggle(Relays.Relay1);

     if (CmdStat != SendStat.Success)
     {
            MessageBox.Show(CmdStat.ToString(), "Error",
                           MessageBoxButtons.OK,
                           MessageBoxIcon.Error);
     }
}
```

Al compilar el programa, si el cable de red no esta conectado al dispositivo y se apreta el boton “RELAY1”, el siguiente cartel deberia aparecer:



Como era de esperar, un cartel de error aparecerá si el comando no puede ser enviado, y mostrara el tipo de error, en este caso “ErrorSend”, indicando que hay un problema en el envío del comando (el cable de red fue desconectado).

El código que utilizamos en el evento “button2_Click“ se explica del siguiente modo:

1. Se crea una variable “CmdStat” del tipo “SendStat”. La misma contendrá el valor retornado por el método “Toggle”.
2. Se envía el comando “Toggle” y el retorno es guardado en “CmdStat” (estado de comando).
3. Se comprueba “CmdStat”, verificando si es distinto a “SendStat.Success”.
4. Si es distinto, hay un error. Entonces se muestra un “MessageBox” con un título de “Error” y un texto de contenido con el valor de “CmdStat” convertido a string.

Eso es todo.

Para el caso de la entrada discreta, procedemos de la misma forma, como se muestra a continuación:

```
private void button1_Click(object sender, EventArgs e)
{
  SendStat CmdStat;
  bool Din1Stat;

  CmdStat = PioBoard.Cmd.Din.CheckIfHigh(DinInput.Din1, out Din1Stat);

  if (CmdStat != SendStat.Success)
  {
      MessageBox.Show(CmdStat.ToString(), "Error",
                      MessageBoxButtons.OK, MessageBoxIcon.Error);
  }

  if (Din1Stat == true)
  {
      textBox1.Text = "1 logico";
  }
  else
  {
      textBox1.Text = "0 logico";
  }
}
```

En el código anterior, se creó la variable “CmdStat” del tipo “SendStat” para almacenar el estado del envío del comando, y luego se verifica si es distinta a “Success”. En caso de ser distinta a “Success”, se mostrara un cartel (MessageBox) para indicar el error.

Adjunto a este documento, se incluye el código del programa con comprobación de errores.

5.11 Próximos Pasos

En esta sección se mostró como crear un programa para enviar comandos al dispositivo STX80XX utilizando la librería de control STX8XXX.DLL en Visual C#.

Como se pudo apreciar, el proceso de creación de un programa práctico para la línea de dispositivos STX80XX es muy rápido y simple. Con práctica, podrá lograr programas más complejos y con mejores representaciones.

El resto del documento, describirá en detalle la librería STX8XXX.DLL para que pueda utilizar como referencia a la hora de hacer sus programas.

6 Librería STX8XXX

En esta sección se describirá la librería STX8XXX.DLL. Se comenzara explicando su organización y luego las clases que la componen. Se brindaran ejemplos de aplicación para facilitar el aprendizaje.

La documentación de la librería se realiza con un orden práctico, de esta forma pretendemos lograr que comprenda el funcionamiento del sistema y no solamente generar un aburrido manual de referencia. Por ello, muchos métodos, clases y estructuras no se documentan en este documento, por no tener uso práctico para el usuario común, sin embargo están documentados "en línea" desde el entorno de programación Microsoft Visual C#.

Los términos clases y objetos, se utilizan de forma indistinta en este documento, dependiendo del contexto, se aplica uno u otro termino.

6.1 Organización

La librería consiste en un set de objetos o clases accesibles desde la clase principal **Stx8xxx**.

La clase **Stx8xxx** contiene métodos, constantes, estructuras y objetos, que son útiles para enviar comandos y recibir datos desde el dispositivo STX80XX. En el documento, cuando "instanciemos" la clase **Stx8xxx**, le daremos el nombre de **PioBoard**.

Instanciar una clase u objeto, significa crearlo en memoria, por lo general con la palabra "new".

Existen otras clases que se pueden "instanciar" de forma independiente a la clase **Stx8xxx** y que llamaremos clases de ayuda, porque brindan apoyo a los programas realizados con la librería STX8XXX. Ejemplo de este tipo de clases, son las clases:

- `UdpStream`
- `UdpStreamHandle`

6.2 Como leer los Métodos

En este documento, los métodos se presentan en el siguiente formato:

Método(Arg1, Arg2): Método de prueba.		
Argumentos	**Tipo**	**Descripción**
Arg1	E	Argumento de entrada.
Arg2	S	Argumento de Salida. Se retorna un valor en la variable Arg2.
Retorno	**Tipo**	**Descripción**
Tipo	S	Operación exitosa.

Donde:

- **Método(Arg1, Arg2)**: Prototipo del método, que incluye nombre de argumentos.
- **Arg1 y Arg2**: Argumentos o parámetros del método. Siempre, observar la naturaleza del argumento (string, entero, clase, etc.).
- **Tipo**: Si es “E” significa entrada. “S” significa salida (en caso de ser un argumento, se entiende que el método retornara un valor en la variable). Si es “E/S” el argumento es de entrada y salida.
- **Retorno**: El retorno del método, es su resultado, puede ser un constante, en ese caso se listan todas las constantes posibles o se especifica el nombre de la enumeración, de lo contrario se le da nombre al retorno con fines descriptivos.

7 Clase Principal Stx8xxx

7.1 Objetivo

Proveer acceso a estructuras de datos, constantes, objetos y métodos, que permiten enviar comandos y recibir datos desde el dispositivo STX80XX.

Mantener datos en variables para configurar la librería.

7.2 Constructores

Inicializa y configura inicialmente la librería STX8XXX.

En el documento, cuando "instanciemos" la clase **Stx8xxx**, le daremos el nombre de **PioBoard**.

Stx8xxx(): Inicializa libreria STX8XXX. Utiliza valores por defecto para inicializar la librería [1].		
Argumentos	**Tipo**	**Descripción**
-		
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Por defecto: IP=192.168.1.81, Port=4950, Password=0, Timeout=2, Retries=1, dispositivo target = STX8081.

Ejemplo:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx();
```

Stx8xxx(string IP, Stx8xxxId TargetDevice): Inicializa libreria STX8XXX. Utiliza valores por defecto para inicializar la librería [1].		
Argumentos	**Tipo**	**Descripción**
IP	E	Dirección IP del dispositivo STX80XX.
TargetDevice	E	Modelo de dispositivo para utilizar con la librería.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Por defecto: Port=4950, Password=0, Timeout=2, Retries=1.

Ejemplo:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("192.168.1.81", Stx8xxxId.STX8081);
```

Stx8xxx(string IP, UInt32 Password, Stx8xxxId TargetDevice): Inicializa libreria STX8XXX. Utiliza valores por defecto para inicializar la librería .		
Argumentos	**Tipo**	**Descripción**
IP	E	Dirección IP del dispositivo STX80XX.
Password	E	Clave o password para utilizar en el dispositivo.
TargetDevice	E	Modelo de dispositivo para utilizar con la librería.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
-		Por defecto: Port=4950, Timeout=2, Retries=1

Ejemplo:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);
```

Stx8xxx(string IP, UInt32 Password, int Port, int Timeout, int Retries, Stx8xxxId TargetDevice): Inicializa libreria STX8XXX. Utiliza valores por defecto para inicializar la librería .		
Argumentos	**Tipo**	**Descripción**
IP	E	Dirección IP del dispositivo STX80XX.
Password	E	Clave o password para utilizar en el dispositivo.
Port	E	Puerto del CSP (Command Server Protocol). Ej: 4950.
Timeout	E	Tiempo a esperar por la respuesta de un comando (en segundos).
Retries	E	Números de intentos de envío de un comando cuando ocurre un Timeout.
TargetDevice	E	Modelo de dispositivo para utilizar con la librería.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
-		

Ejemplo:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("192.168.1.81", 0,
                       (int)Stx8xxxUdpPorts.CmdServer,
                       2, 1, Stx8xxxId.STX8081);
```

Stx8xxx(string IP_Or_Hostname, UInt32 Password, Stx8xxxId TargetDevice, Stx8xxxAddressTypeFormat Type): Inicializa libreria STX8XXX. Utiliza valores por defecto para inicializar la librería.		
Argumentos	**Tipo**	**Descripción**
IP_Or_Hostname	E	Dirección IP o hostname del dispositivo STX80XX.
Password	E	Clave o password para utilizar en el dispositivo.
TargetDevice	E	Modelo de dispositivo para utilizar con la librería.
Type	E	Tipo de dirección a utilizar: Hostname = Nombre de host. IP = Dirección IP. Unknown= Auto detectar.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Este método permite utilizar un nombre en vez de una dirección IP.

Ejemplo:

El siguiente código inicializa la librería para utilizar un dispositivo conectado a la dirección “myhome.net” en vez a una dirección IP estática como “192.168.1.81”.

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("myhome.net", 0,
                       Stx8xxxId.STX8081,
                       Stx8xxxAddressTypeFormat.Hostname);
```

Notas:

Stx8xxxUdpPorts.CmdServer: Contiene el numero de puerto UDP del Command Server Protocol.

7.3 Objetos Miembros

La clase contiene dos objetos:

1. **Cmd** : Contiene métodos y objetos para enviar comandos al dispositivo y otras utilidades.
2. **Globals**: Provee información global, métodos de configuración y métodos varios de la librería.

7.4 Objeto PioBoard.Cmd

Contiene objetos que comandan el dispositivo. Comúnmente se los emplea para utilizar las características disponibles del dispositivo.

Objetos disponibles en "Cmd"	
Objeto	**Descripción**
Board	Métodos varios para controlar el dispositivo STX80XX en general.
BoardConfig	Configuración del dispositivo.
BoardInfo	Obtiene información del dispositivo.
Count	Control de entradas de contadores (COUNTx).
Din	Control de entradas digitales o discretas (DINx).
Dout	Control de salidas digitales o discretas (DOUTx)
Lcd	Control de display LCD.
NetHmi	Permite enviar y recibir paquetes NetHMI del dispositivo en modo PLC.
PWM	Control de salidas PWM (PWMx).
Relay	Control de salidas RELAY (RELAYx).
Udp	Interface para enviar datos UDP del dispositivo en modo PLC.
Vin	Control de entradas analógicas (VINx).
Vout	Control de salida analógica (VOUTx).

El acceso a los objetos en "Cmd", se explicará con detalles más adelante en el documento, ya que desde aquí es posible enviar los comandos al dispositivo STX80XX.

7.5 Objeto PioBoard.Globals

Contiene objetos y métodos que obtienen información de la librería y modifican su funcionamiento.

Objetos disponibles en "Globals"	
Objeto	**Descripción**
CmdClient	Objeto de "bajo-nivel" para enviar y ejecutar comandos en dispositivo STX80XX utilizando el protocolo **CSP**.
LibInfo	Suministra información de la librería y permite configurar parámetros globales.

Metodos disponibles en "Globals"	
Objeto	**Descripción**
SetCmdServerConnection	Especifica parámetros de la conexión entre el dispositivo STX80XX y la PC, utilizando el protocolo **CSP**. Parámetros accesibles: IP, Port, Timeout y Retries.
SetCmdServerTimeout	Especifica el tiempo de "timeout"o de espera de respuesta al enviar un comando.
SetCmdServerPassword	Especifica el password a utilizar cuando se envíen comandos al STX80XX.
GetCmdServerPassword	Obtiene el password actual utilizado por esta librería.
SetCmdServerProtoInfo	Especifica información utilizada en el protocolo CSP.
GetCmdServerProtoInfo	Obtiene información utilizada en el protocolo CSP.

7.5.1 Métodos del Objeto PioBoard.Globals

void SetCmdServerConnection(string IP): Especifica parámetros de la conexión entre el dispositivo STX80XX y la PC, utilizando el protocolo **CSP**.		
Argumentos	**Tipo**	**Descripción**
IP	E	Dirección IP del dispositivo STX80XX.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Permite cambiar parámetros de conexión sin volver a instanciar la clase.

Ejemplo:

Supongamos que tenemos dos dispositivos, uno con dirección "192.168.1.81" y otro con dirección "192.168.1.82". Si la librería estaba configurada para enviar comandos al primer dispositivo, podemos especificar con este método que posteriores comandos sean enviados al segundo dispositivo:

```
PioBoard.Globals.SetCmdServerConnection("192.168.1.82");
```

void SetCmdServerConnection(string IP, int Port): Especifica parámetros de la conexión entre el dispositivo STX80XX y la PC, utilizando el protocolo **CSP**.		
Argumentos	**Tipo**	**Descripción**
IP	E	Dirección IP del dispositivo STX80XX.
Port	E	Puerto UDP donde el dispositivo STX80XX escucha conexiones del protocolo CSP.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Permite cambiar parámetros de conexión sin volver a instanciar la clase.

Ejemplo:

En este caso, se cambio el puerto por defecto del protocolo CSP, de 4950 a 6013. Para que esto funcione, el dispositivo también debería también cambiar el puerto del CSP.

```
PioBoard.Globals.SetCmdServerConnection("192.168.1.81", 6013);
```

Si se desea utilizar el puerto por defecto del CSP, utilizar:

```
PioBoard.Globals.SetCmdServerConnection("192.168.1.81",  (int) Stx8xxxUdpPorts.CmdServer);
```

void SetCmdServerConnection(string IP, int Port, double Timeout): Especifica parámetros de la conexión entre el dispositivo STX80XX y la PC, utilizando el protocolo **CSP**.		
Argumentos	**Tipo**	**Descripción**
IP	E	Dirección IP del dispositivo STX80XX.
Port	E	Puerto UDP donde el dispositivo STX80XX escucha conexiones del protocolo CSP.
Timeout	E	Tiempo de espera para la respuesta de un comando, en segundos.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Permite cambiar parámetros de conexión sin volver a instanciar la clase.

Ejemplo:

Si se desea esperar 15 segundos por la respuesta de un comando, utilizar:

```
PioBoard.Globals.SetCmdServerConnection("192.168.1.81", (int) Stx8xxxUdpPorts.CmdServer, 15);
```

void SetCmdServerConnection(string IP, int Port, double Timeout): Especifica parámetros de la conexión entre el dispositivo STX80XX y la PC, utilizando el protocolo **CSP**.		
Argumentos	**Tipo**	**Descripción**
IP	E	Dirección IP del dispositivo STX80XX.
Port	E	Puerto UDP donde el dispositivo STX80XX escucha conexiones del protocolo CSP.
Timeout	E	Tiempo de espera para la respuesta de un comando, en segundos.
Tries	E	Intentos para enviar un comando, en caso de no obtener respuesta.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Permite cambiar parámetros de conexión sin volver a instanciar la clase.

Ejemplo:

El siguiente ejemplo, le dice a la librería que espere 5 segundos por la respuesta de cada comando enviado al dispositivo, en caso que no exista respuesta (timeout o error), intentarlo 3 veces.

```
PioBoard.Globals.SetCmdServerConnection("192.168.1.81", (int) Stx8xxxUdpPorts.CmdServer, 5, 3);
```

void SetCmdServerTimeout(double Timeout): Especifica el tiempo de espera para la respuesta de un comando cuando se utiliza el protocolo **CSP**.		
Argumentos	**Tipo**	**Descripción**
Timeout	E	Tiempo de espera para la respuesta de un comando, en segundos.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Permite cambiar parámetro Timeout de la conexión sin volver a instanciar la clase.

Ejemplo:

Normalmente se utiliza para cambiar el tiempo de espera para la respuesta de un comando que pueda tardar mucho en ejecutarse o si la conexión es muy lenta. En el siguiente ejemplo, se esperan 30.5 segundos como máximo por la respuesta cuando se envíe un comando al dispositivo:

```
PioBoard.Globals.SetCmdServerTimeout(30.5);
```

void SetCmdServerRetries(int Tries): Especifica el numero de re-intentos para intentar nuevamente enviar un comando cuando ocurre un Timeout en la espera de respuesta de un comando al utilizar el protocolo **CSP**.		
Argumentos	**Tipo**	**Descripción**
Tries	E	Intentos para enviar un comando, en caso de no obtener respuesta.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Permite cambiar parámetro Tries de la conexión sin volver a instanciar la clase.
2		Tenga en cuenta que a mayor re-intentos, la librería tardará más tiempo en retornar de un comando enviado cuando no exista respuesta.

Ejemplo:

Realizar 5 re-intentos cuando no se obtiene respuesta de un comando enviado al dispositivo.

```
PioBoard.Globals.SetCmdServerRetries(5);
```

void SetCmdServerPassword(UInt32 Password): Especifica el password o clave que utilizara la librería al enviar comandos al dispositivo cuando se utiliza el protocolo **CSP**.		
Argumentos	**Tipo**	**Descripción**
Password	E	Clave. Consiste en un número entero y positivo de 32-bits. Por defecto el dispositivo utiliza la cave 0.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Es importante que usted conozca la clave de su dispositivo STX80XX antes de enviar comandos.

Ejemplo:

Si la clave de su dispositivo es “82839”, usted puede especificarla en la librería de la siguiente forma:

```
PioBoard.Globals.SetCmdServerPassword(82839);
```

UInt32 GetCmdServerPassword(): Retorna el password utilizado por la librería al enviar comandos al dispositivo cuando se utiliza el protocolo **CSP**.

Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
Password	S	Password de 32-bits utilizado por la librería.
Notas		**Descripción**
-		

Ejemplo:

En el siguiente código se guarda el password utilizado en una variable entera de 32-bits llamada "MyPassword".

```
UInt32 MyPassword;

MyPassword = PioBoard.Globals.GetCmdServerPassword();
```

void SetCmdServerProtoInfo(byte ProtoInfo): Especifica informacion del del protocolo **CSP** usado en todas las transacciones.

Argumentos	**Tipo**	**Descripción**
ProtoInfo	E	Información del protocolo. Por ejemplo flags y versión del protocolo.
Retorno	**Tipo**	**Descripción**
-	-	
Notas		**Descripción**
-		

Ejemplo:

No hay ejemplo práctico para el usuario de este método.

byte GetCmdServerProtoInfo():Retorna informacion del del protocolo **CSP** usado en todas las transacciones.

Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
ProtoInfo	S	Información del protocolo. Por ejemplo flags y versión del protocolo.
Notas		**Descripción**
-		

Ejemplo:

No hay ejemplo práctico para el usuario de este método.

7.5.2 Objeto PioBoard.Globals.LibInfo

Obtiene información de la librería STX8XXX.

Contiene:

1. Información de la librería: Versión, autor, etc.
2. Información de parámetros del dispositivo STX80XX: Offssets, valores del ADC, etc.

Método para obtener Información de la Librería:

LibraryInfoCommon GetLibraryCommonInformation(): Retorna informacion de la librería STX8XXX.DLL.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
LibraryInfoCommon	S	Información de la librería en una estructura LibraryInfoCommon.
Notas		**Descripción**
-		

Ejemplo:

```
LibraryInfoCommon LibInfo;

LibInfo = PioBoard.Globals.LibInfo.GetLibraryCommonInformation();
```

La estructura LibraryInfoCommon tiene los siguientes campos:

struct LibraryInfoCommon: Estructura con informacion general de la libreria.			
Campos	**Tipo**	**Valor**	**Descripción**
LibName	string	-	Nombre de librería.
LibVersion	int	-	Versión del software de la librería
AuthorName	string	-	Nombre del autor de la librería.
AuthorMail	string	-	Dirección de e-mail del autor.
AuthorWeb	string	-	Dirección web del autor.
Manufacturer	string	-	Fabricante de la librería.

Información de parámetros del dispositivo:

Es una estructura que contiene información sobre el funcionamiento del hardware del dispositivo STX80XX, la misma tiene valores asignados por fábrica. No debería modificarse a menos que se esté seguro de lo que se está haciendo. Es accesible mediante:

```
PioBoard.Globals.LibInfo.PioBoardInfo
```

Y PioBoardInfo es una estructura del tipo: PioBoardInformation .

Los campos de la estructura están documentados en la documentación en línea y puede obtenerse la información desde allí.

8 Objetos para Enviar Comandos

Esta sección es la más práctica del documento, aquí se describirá como enviar los comandos que permiten utilizar las características del dispositivo STX80XX.

Los ejemplos suponen que usted ya inicializo la librería STX8XXX.DLL correctamente, si no lo hizo, lea las secciones anteriores.

Los objetos para enviar los comandos, son accesibles desde el objeto “Cmd”:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx();

// Se accede a los objetos de “Cmd”:
PioBoard.Cmd.
```

```
Board
BoardConfig
BoardInfo
Count
Din
Equals
GetHashCode
GetType
PWM
Relay
```

```
// Se envia el comando para "cerrar" el RELAY1.
PioBoard.Cmd.Relay.Close(Relays.Relay1);
```

Antes de continuar, explicaremos algunas estructuras y enumeraciones útiles, que son utilizadas con frecuencia al momento de enviar comandos al dispositivo STX80XX.

8.1 Enumeración SendStat

La enumeración “**SendStat**” contiene constantes que son devueltas por la mayoría de los comandos que se utilizan para comandar el dispositivo STX80XX. Hay algunas excepciones, que se comentarán en su debida sección. Se suele denominar al proceso de enviar comando y recibir una respuesta, como “**transacción**”.

Estas constantes permiten conocer el estado de retorno al enviar y ejecutar un comando en el dispositivo STX80XX. Un retorno exitoso, comando ejecutado correctamente, debería devolver ***SendStat.Success***, de lo contrario, si no se devuelve ese valor, existió algún error.

A continuación se lista la definición de la enumeración “**SendStat**” con sus posibles valores:

enum SendStat : int: Enumeracion que representa el estado de la transaccion de un comando.			
Campos	**Tipo**	**Valor (dec)**	**Descripción**
Success	int	0	Retorno exitoso de función.
ErrorTimeout	int	-1	Tiempo de espera por respuesta de comando expiro.
ErrorUnknown	int	-2	Otro error, mirar código fuente de librería.
ErrorIncompleteHeaders	int	-3	Paquete incompleto (headers/cabeceras no recibidas).
ErrorIncompleteOutput	int	-4	Headers recibidos pero recepción de datos de salida de comando incompletos.
ErrorSend	int	-5	Se ha retornado un error cuando se intento acceder al socket para transmitir datos.
ErrorSendIncomplete	int	-6	Transmisión incompleta.
ErrorCmdRtnStat	int	-7	CRP (Command Return Packet) fue recibido, pero el campo STAT0 contiene un código de error (error en la ejecución de comando). Comprobar retorno de comando, campos STAT0 y STAT1 por mas información.
ErrorRtnProtocolVersion	int	-8	Versión de protocolo CSP retornado desde la STX80XX no está soportada por esta librería.
ErrorProtocolVerUnsupported	int	-9	Versión de protocolo CSP utilizado por esta librería, no está soportada por la STX80XX.
ErrorPasswordIncorrect	int	-10	El comando no fue ejecutado debido a que el password es incorrecto. Comprobar password!.
ErrorTypeNotExist	int	-11	El tipo de comando para ejecutar en la STX80XX no existe. Comprobar campo TYPE.
ErrorCmdUnknown	int	-12	El comando que se está tratando de ejecutar no existe. Comprobar el campo ID.

enum SendStat : int: Enumeracion (continuacion).

Campos	Tipo	Valor (dec)	Descripción
ErrorCmdReturnError	int	-13	El comando ejecutado ha retornado un error. Quizás el tipo de error fue retornado en la salida de datos del comando (bytes del campo R[0]…R[N]). Mire la descripción del comando para más información.
ErrorRxUnknown	int	-14	Error desconocido en la recepción del paquete. Mirar código fuente.
ErrorIncompleteArgs	int	-15	Argumentos del comando enviado a la STX80XX están incompletos. Comprobar valor de campo ARGSIZE y campos de datos ARGS.
ErrorWrongArguments	int	-16	Los argumentos del comando enviado son incorrectos
ErrorCmdUnsopportedInPLC	int	-17	El comando que está tratando de ejecutar no está soportado cuando el modo PLC está activo en el dispositivo. Comprobar que el ID del comando sea un valor entre 0 y 19.
ErrorCmdUnsopportedInBootloader	int	-18	El comando que está tratando de ejecutar no está soportado cuando el modo Bootloader está activo en el dispositivo. Comprobar el ID del comando o cambiar el modo de funcionamiento del dispositivo a PLC o DAQ.
ErrorOnSocketBindOrIsBusy	int	-19	Otro recurso está intentando acceder al socket de conexión al mismo tiempo.
ErrorResolvHostnameIP	int	-20	No se puede obtener la dirección IP del Hostname. Comprobar conexión internet, nombre de Hostname valido y configuración DNS.

Valores de retorno distinto a “0” o “Success” desde el dispositivo significan error.

8.2 Objeto PioBoard.Cmd.Dout

Controla las salidas digitales del dispositivo STX80XX. A continuación métodos y constates genéricas para controlar las salidas digitales del dispositivo. La cantidad de salidas y tipo de salida (relé, transistor, etc.) disponible depende del modelo de dispositivo. Consulte hoja de datos.

Los métodos del objeto "Dout" son genéricas, es decir permiten controlar cualquier salida digital que tenga dos estados, ON y OFF. Por activar se dice a la acción de colocar un "1" como estado lógico y por desactivar se entiende como colocar un "0" como estado lógico. Esto puede tener distintos sentidos dependiendo de la naturaleza de la salida, por ejemplo ya sea un relé normal abierto o normal cerrado. Es posible utilizar funciones mas especificas cuando utilice relés, como las descriptas en la página 52.

Nota: El soporte "Dout" se introduce en versiones nuevas de firmware del dispositivo, se recomienda tener actualizado el firmware a la última versión.

8.2.1 Métodos

SendStat SetOn(Douts Dout): Activa las salidas seleccionadas en el dispositivo.		
Argumentos	**Tipo**	**Descripción**
Dout	E	Salida o salidas seleccionadas para activarse.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo básico:

```
// Activar solamente DOUT1.
PioBoard.Cmd.Dout.SetOn(Douts.Dout1);

// Activar DOUT2, DOUT4 y DOUT7 al mismo tiempo.
// Notar que con el operador "|" podemos especificar varios DOUTs al mismo tiempo.
PioBoard.Cmd.Dout.SetOn(Douts.Dout2 | Douts.Dout4 | Douts.Dout7);

// Activar todos los DOUT al mismo tiempo.
PioBoard.Cmd.Dout.SetOn(Douts.All);
```

Ejemplo con comprobación de errores:

```
// Crear una variable "CmdStat" del tipo "SendStat" para guardar el estado de una transaccion.
SendStat CmdStat = SendStat.Success;

// Activar DOUT1 y guardar estado de transaccion en variable "CmdStat".
CmdStat = PioBoard.Cmd.Dout.SetOn(Douts.Dout1);
```

```
// Comprobar si la transaccion o el comando enviado al dispositivo fue exitoso.
if (CmdStat != SendStat.Success)
{
    // NO !... Error!

    // Abrir ventana de dialogo (MessageBox) y mostrar el codigo del error retornado.
    MessageBox.Show(CmdStat.ToString(), "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Más adelante se describirá la enumeración “Douts”.

SendStat SetOff(Douts Dout): Desactiva las salidas seleccionadas en el dispositivo.

Argumentos	**Tipo**	**Descripción**
Dout	E	Salida o salidas seleccionadas para desactivarse.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Desacrivar DOUT3 del dispositivo.
PioBoard.Cmd.Dout.SetOff(Douts.Dout3);
```

SendStat Toggle(Douts Dout): Invierte el estado las salidas seleccionadas en el dispositivo. Si la salida estaba desactivada, la ejecucion de este metodo activará la salida y viceversa.

Argumentos	**Tipo**	**Descripción**
Dout	E	Salida o salidas seleccionados para invertir.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Cambiar estado de DOUT6.
PioBoard.Cmd.Dout.Toggle(Douts.Dout6);
```

SendStat GetState(out UInt32 DoutStat): Obtiene el estado de todas las salidas del dispositivo.		
Argumentos	**Tipo**	**Descripción**
DoutStat	S	Estado de todas las salidas. Cada bit representa el estado de una salida.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Esta función puede ser útil para conocer el estado de las salidas del dispositivo.
2		Ejemplo, si las salidas 3 y 5 están activadas, el retorno de la variable utilizada para almacenar el estado será 00010100.

Ejemplo:

```
// Variable donde se almacenará el estado de las salidas.
UInt32 DoutStatus;

// Obtener estado de las salidas.
PioBoard.Cmd.Dout.GetState(out DoutStatus);
```

bool CheckIfSetOn(UInt32 DoutState, Douts Dout): Comprueba si una salida esta activada.		
Argumentos	**Tipo**	**Descripción**
DoutState	E	Estado de las salidas del dispositivo. Cada bit representa el estado de una salida.
Dout	E	Nombre de salida a comprobar.
Retorno	**Tipo**	**Descripción**
bool	S	Estado del salida. True significa activada y False significa desactivada.
Notas		**Descripción**
-		

Ejemplo:

```
// Variable donde se almacenará el estado de las salidas.
UInt32 DoutStatus;

// Obtener estado de las salidas.
PioBoard.Cmd.Dout.GetState(out DoutStatus);

// Comprobar si el DOUT5 esta activada.
if (PioBoard.Cmd.Dout.CheckIfSetOn(DoutStatus, Douts.Dout5) == true)
{
   // DOUT5 activada!.
   // Código ...
}
```

```
else
{
   // DOUT5 desactivada.
   // Código ...
}
```

8.2.2 Enumeración Douts

enum Douts : byte : Enumeracion que representa las salidas disponibles. La cantidad depende del modelo de dispositivo. Consulte hoja de datos.

Campos	Tipo	Valor (bin)	Descripción
Dout1	uint	00000000 00000001	Representa el DOUT1.
Dout2	uint	00000000 00000010	Representa el DOUT2.
Dout3	uint	00000000 00000100	Representa el DOUT3.
Dout4	uint	00000000 00001000	Representa el DOUT4.
Dout5	uint	00000000 00010000	Representa el DOUT5.
Dout6	uint	00000000 00100000	Representa el DOUT6.
Dout7	uint	00000000 01000000	Representa el DOUT7.
Dout8	uint	00000000 10000000	Representa el DOUT8.
Dout9	uint	00000001 00000000	Representa el DOUT9.
…	uint	…	Representa el DOUT ...
Dout16	uint	10000000 00000000	Representa el DOUT16.
…	uint	…	Representa el DOUT ...
Dout32	uint	10000000 00000000 00000000 00000000	Representa el DOUT32.

8.3 Objeto PioBoard.Cmd.Relay

Controla los relés RELAYx del dispositivo STX80XX.

8.3.1 Métodos

SendStat Close(Relays Relay): Cierra relays seleccionados en el dispositivo.		
Argumentos	**Tipo**	**Descripción**
Relay	E	Relay o Relays seleccionados para cerrarse.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo básico:

```
// Cerrar solamente RELAY1.
PioBoard.Cmd.Relay.Close(Relays.Relay1);

// Cerrar RELAY2, RELAY4 y RELAY7 al mismo tiempo.
// Notar que con el operador “|” podemos especificar varios RELAYs al mismo tiempo.
PioBoard.Cmd.Relay.Close(Relays.Relay2 | Relays.Relay4 | Relays.Relay7);

// Cerrar todos los RELAY al mismo tiempo.
PioBoard.Cmd.Relay.Close(Relays.All);
```

Ejemplo con comprobación de errores:

```
// Crear una variable “CmdStat” del tipo “SendStat” para guardar el estado de una transaccion.
SendStat CmdStat = SendStat.Success;

// Cerrar RELAY1 y guardar estado de transaccion en variable “CmdStat”.
CmdStat = PioBoard.Cmd.Relay.Close(Relays.Relay1);

// Comprobar si la transaccion o el comando enviado al dispositivo fue exitoso.
if (CmdStat != SendStat.Success)
{
    // NO !... Error!

    // Abrir ventana de dialogo (MessageBox) y mostrar el codigo del error retornado.
    MessageBox.Show(CmdStat.ToString(), "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Más adelante se describirá la enumeración “Relays”.

SendStat Open(Relays Relay): Abre los relays seleccionados en el dispositivo.		
Argumentos	**Tipo**	**Descripción**
Relay	E	Relay o Relays seleccionados para abrirse.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Abrir RELAY3 del dispositivo.
PioBoard.Cmd.Relay.Open(Relays.Relay3);
```

SendStat Toggle(Relays Relay): Invierte el estado de los relays seleccionados en el dispositivo. Si el rele estaba abierto, la ejecucion de este metodo cerrara el rele y viceversa.		
Argumentos	**Tipo**	**Descripción**
Relay	E	Relay o Relays seleccionados para invertir.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Cambiar estado de RELAY6.
PioBoard.Cmd.Relay.Toggle(Relays.Relay6);
```

SendStat GetState(out UInt32 RelayStat): Obtiene el estado de todos los reles del dispositivo.		
Argumentos	**Tipo**	**Descripción**
RelayStat	S	Estado todos los relés. Cada bit representa el estado de un relé.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Esta función puede ser útil para conocer el estado de los relays del dispositivo.
2		Ejemplo, si los relays 3 y 5 están cerrados, el retorno de la variable de salida será 00010100.

Ejemplo:

```
// Variable donde se almacenara el estado de los relays.
UInt32 RelayStatus;

// Obtener estado de los relays.
PioBoard.Cmd.Relay.GetState(out RelayStatus);
```

bool CheckIfClose(UInt32 RelayState, Relays Relay): Comprueba si un rele esta cerrado (activado).		
Argumentos	**Tipo**	**Descripción**
RelayState	E	Estado de los relés del dispositivo. Cada bit representa el estado de un relé.
Relay	E	Nombre del relé a comprobar.
Retorno	**Tipo**	**Descripción**
bool	S	Estado del relé. True significa relé activado y False significa relé abierto.
Notas		**Descripción**
-		

Ejemplo:

```
// Variable donde se almacenara el estado de los relays.
UInt32 RelayStatus;

// Obtener estado de los relays.
PioBoard.Cmd.Relay.GetState(out RelayStatus);

// Comprobar si el RELAY5 esta activado.
if (PioBoard.Cmd.Relay.CheckIfClose(RelayStatus, Relays.Relay5) == true)
{
   // RELAY5 Cerrado!.
   // Código ...
}
else
{
   // RELAY5 Abierto.
   // Código ...
}
```

8.3.2 Enumeración Relays

enum Relays : byte : Enumeracion que representa los relays disponibles.

Campos	Tipo	Valor (bin)	Descripción
Relay1	uint	00000000 00000001	Representa el RELAY1.
Relay2	uint	00000000 00000010	Representa el RELAY2.
Relay3	uint	00000000 00000100	Representa el RELAY3.
Relay4	uint	00000000 00001000	Representa el RELAY4.
Relay5	uint	00000000 00010000	Representa el RELAY5.
Relay6	uint	00000000 00100000	Representa el RELAY6.
Relay7	uint	00000000 01000000	Representa el RELAY7.
Relay8	uint	00000000 10000000	Representa el RELAY8.
Relay9	uint	00000001 00000000	Representa el RELAY9.
…	uint	…	Representa el RELAY ...
Relay16	uint	10000000 00000000	Representa el RELAY16.
…	uint	…	Representa el RELAY ...
Relay32	uint	10000000 00000000 00000000 00000000	Representa el RELAY32.

8.4 Objeto PioBoard.Cmd.Din

Comandos para leer el estado de las entradas digitales o discretas DINx del dispositivo STX80XX.

8.4.1 Métodos

SendStat ReadAll(out UInt32 DinState): Obtiene el estado de todos las entradas discretas.		
Argumentos	**Tipo**	**Descripción**
DinState	S	Estado de todas las entradas discretas. Cada bit representa el estado de una entrada discreta.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Ejemplo, si las entradas DIN3 y DIN5 están polarizadas, el retorno de la variable de salida será 00010100.

Ejemplo:

```
// Variable donde se almacenara el estado de
// las entradas discretas.
UInt32 DinStatus;

// Obtener el estado de todas las entradas discretas.
PioBoard.Cmd.Din.ReadAll(out DinStatus);
```

SendStat CheckIfHigh(DinInput Din, out bool DinStat): Comprueba si una entrada discreta tiene un “uno logico” o esta polarizada.		
Argumentos	**Tipo**	**Descripción**
Din	E	Entrada discreta a comprobar.
DinStat	S	Estado de la entrada discreta (si es “true” la entrada fue polarizada).
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Este método envía un comando para leer la entrada.

Ejemplo:

```
// Variable que almacenara el estado de la entrada DIN1.
bool Din1Stat;

// Obtener valor de entrada DIN1 en variable Din1Stat.
PioBoard.Cmd.Din.CheckIfHigh(DinInput.Din1, out Din1Stat);
```

```
// Codigo de ejemplo:
if (Din1Stat == true)
{
    // Entrada DIN1 en nivel alto.
    // ...
 }
```

bool CheckIfHigh(UInt32 DinState, DinInput Din): Comprueba si una entrada discreta tiene un "uno logico" o esta polarizada.		
Argumentos	**Tipo**	**Descripción**
DinStat	E	Estados de la entradas discretas (ver método ReadAll() para más información).
Din	E	Entrada discreta a comprobar.
Retorno	**Tipo**	**Descripción**
bool	S	Estado de la entrada discreta (si es "true" la entrada fue polarizada).
Notas		**Descripción**
1		Este método **no envía un comando** para comprobar la entrada discreta.

Ejemplo:

```
// Variable donde se almacenara el estado de
// las entradas discretas.
UInt32 DinState;

// Obtener el estado de las entradas discretas.
PioBoard.Cmd.Din.ReadAll(out DinState);

// Comprobar el valor de entrada DIN1 utilizando la variable DinState previamente obtenida.
if (PioBoard.Cmd.Din.CheckIfHigh(DinState, DinInput.Din1) == true)
{
    // Entrada discreta DIN1 en alto.
    // Codigo...
}

// Comprobar el valor de entrada DIN2 utilizando la variable DinState previamente obtenida.
if (PioBoard.Cmd.Din.CheckIfHigh(DinState, DinInput.Din2) == true)
{
    // Entrada discreta DIN2 en alto.
    // Codigo...
}

// Comprobar el valor de entrada DIN3 utilizando la variable DinState previamente obtenida.
if (PioBoard.Cmd.Din.CheckIfHigh(DinState, DinInput.Din3) == true)
{
    // Entrada discreta DIN3 en alto.
    // Codigo...
}
```

Notar que con este método se pueden comprobar múltiples entradas, enviando solo un comando.

SendStat InformChanges(bool Inform): Este metodo configura a la PioBoard para enviar paquetes UDP-STREAM cuando el estado actual de las entradas discretas DIN difiere del ultimo estado leido por el usuario.		
Argumentos	**Tipo**	**Descripción**
Inform	E	Estados de la entradas discretas (ver método ReadAll() para más información).
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Receibir paquetes UDP-STREAM cuando las entradas
// discretas DIN cambien de estado.
PioBoard.Cmd.Din.InformChanges(true);
```

En secciones posteriores se brindara más información sobre cómo manejar paquetes UDP-STREAM.

8.4.2 Enumeración DinInput

enum DinInput : uint: Enumeracion que representa las entradas discretas disponibles en la STX80XX.			
Campos	**Tipo**	**Valor (bin)**	**Descripción**
Din1	uint	00000001	Representa la entrada DIN1.
Din2	uint	00000010	Representa la entrada DIN2.
Din3	uint	00000100	Representa la entrada DIN3.
Din4	uint	00001000	Representa la entrada DIN4.
Din5	uint	00010000	Representa la entrada DIN5.
Din6	uint	00100000	Representa la entrada DIN6.
Din7	uint	01000000	Representa la entrada DIN7.
Din8	uint	10000000	Representa la entrada DIN8.
…	uint	…	
Din32	uint	10000000 00000000 00000000 00000000	Representa la entrada DIN32.

8.5 Objeto PioBoard.Cmd.PWM

Comandos para controlar las salidas PWMx del dispositivo STX80XX.

8.5.1 Métodos

SendStat Frequency(UInt16 Frequency): Establece la frecuencia del PWM en todos los canales.		
Argumentos	**Tipo**	**Descripción**
Frequency	E	Frecuencia del PWM en Hz. Valor mínimo 1 Hz. Máximo, ver hoja de datos.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Al cambiar la frecuencia, el DutyCycle de las salidas PWM debería modificarse. Por lo tanto, para mantenerlo en un valor fijo, se debe actualizar el DutyCycle luego de cambiar la frecuencia.

Ejemplo:

```
// Establecer la frecuencia de salidas PWM en 230 Hz.
PioBoard.Cmd.PWM.Frequency(230);
```

SendStat DutyCycle(PwmCh Pwm, float DutyCycle): Establece el ciclo de trabajo (DutyCycle) de una o varias salidas PWM.		
Argumentos	**Tipo**	**Descripción**
Pwm	E	Salida PWM que se desea modificar el DutyCycle.
DutyCycle	E	Ciclo de trabajo. Rango 0% a 100%, resolución 0.1 %.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Establecer DutyCycle al 35.5% en salida PWM1.
PioBoard.Cmd.PWM.DutyCycle(PwmCh.PWM1, 35.5f);

// Establecer DutyCycle al 50.0% en salidas PWM1 y PWM2.
PioBoard.Cmd.PWM.DutyCycle(PwmCh.PWM1and2, 50);
```

8.5.2 Enumeración PwmCh

enum PwmCh : byte: Enumeracion que representa las salidas PWM disponibles en la STX80XX.

Campos	Tipo	Valor (bin)	Descripción
PWM1	byte	01	Representa la salida PWM1.
PWM2	byte	10	Representa la salida PWM2.
None	byte	00	No selecciona ninguna salida PWM (sin efecto).
All	byte	11	Representa todas las salidas PWM.
PWM1and2	byte	11	Representa las salidas PWM1 y PWM2.

© 2015 Slicetex Electronics

8.6 Objeto PioBoard.Cmd.Vout

Comandos para controlar las salidas analógicas VOUTx.

Consulte hoja de datos del dispositivo para rangos de tensión y resoluciones disponibles en salidas analógicas.

8.6.1 Métodos de Rangos en Salidas

SendStat SetRange(VoutCh Vout, VoutVoltRange Range): Especifica rango de voltage en salida		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Range	E	Rango de voltaje. Ver enumeración VoutVoltRange.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1	-	Algunos dispositivos pueden requerir que además configure el rango de voltaje de salida con un jumper. Consulte hoja de datos del dispositivo.

Ejemplo:

```
// Especificar rango de 0-10V en la salida analógica VOUT1.
 PioBoard.Cmd.Vout.SetRange(VoutCh.Vout1, VoutVoltRange.Unipolar_10V);
```

VoutVoltRange GetRange(VoutCh Vout): Obtiene el rango de voltaje configurado en la salida.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Retorno	**Tipo**	**Descripción**
VoutVoltRange	S	Rango de voltaje de la salida. Ver enumeración VoutVoltRange.
Notas		**Descripción**
-	-	

Ejemplo:

```
// Obtener rango de la salida analógica VOUT1.
 VoutVoltRange  Range = PioBoard.Cmd.Vout.GetRange(VoutCh.Vout1);
```

8.6.2 Métodos de Resoluciones en Salidas

int GetResolution(VoutCh Vout): Retorna la resolución en bits de la salida analógica.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Retorno	**Tipo**	**Descripción**
Bits	S	Valor numérico de la resolución utilizada por la salida analógica (ej: 8,10,12,etc)
Notas		**Descripción**
1	-	Consulte la hoja de datos de su dispositivo para mayor resolución disponible.
2	-	Si la resolución de la salida VOUT1 es 10-bits, el retorno será el valor 10.

Ejemplo:

```
// Obtener resolución de la salida analógica VOUT1.
int Res = PioBoard.Cmd.Vout.GetResolution(VoutCh.Vout1);
```

int GetResolutionMax(VoutCh Vout): Retorna el maximo valor binario admisible por la salida analógica.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Retorno	**Tipo**	**Descripción**
Value	S	Valor numérico.
Notas		**Descripción**
1	-	Ejemplo, si la resolución de la salida es 10-bits, el retorno será 2^10 -1 = 1023.

Ejemplo:

```
// Obtener maximo valor admisible por la salida analógica VOUT1.
int Max = PioBoard.Cmd.Vout.GetResolutionMax(VoutCh.Vout1);
```

int GetResolutionMid(VoutCh Vout): Retorna el punto medio binario de acuerdo a la resolución naxima.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Retorno	**Tipo**	**Descripción**
Value	S	Valor numérico.
Notas		**Descripción**
1	-	Ejemplo, si la resolución de la salida es 10-bits, el retorno será 510.
2	-	Es un valor útil por si se necesita colocar la salida a una tensión en punto medio.

Ejemplo:

```
// Obtener valor de punto medio de la salida analógica VOUT1.
 int Mid = PioBoard.Cmd.Vout.GetResolutionMid(VoutCh.Vout1);
```

int GetZeroValue(VoutCh Vout): Retorna el valor en binario para producir un voltage cero en la salida.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Retorno	**Tipo**	**Descripción**
Value	S	Valor numérico.
Notas		**Descripción**
1	-	En algunos modelos o salidas analógicas este valor puede no estar disponible. Por ejemplo en salidas que no generan 0 Volts.

Ejemplo:

```
// Obtener valor binario para producir un voltaje de 0 Volts en salida analógica VOUT1.
 int Zero = PioBoard.Cmd.Vout.GetZeroValue(VoutCh.Vout1);
```

8.6.3 Métodos para Escribir en Salidas

SendStat VoutWrite(VoutCh Vout, UInt16 Value): Ecribe un valor digital en la salida analogica.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Value	E	Valor digital a escribir en salida.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Escribir el valor digital 526 en la salida analógica.
PioBoard.Cmd.Vout.VoutWrite((VoutCh.Vout1, 526);

// Escribir el valor digital maximo en la salida analogica.
PioBoard.Cmd.Vout.VoutWrite(VoutCh.Vout1, \
        (UInt16) PioBoard.Cmd.Vout.GetResolutionMax(VoutCh.Vout1));
```

SendStat Vout(VoutCh Vout, float Voltage): Ecribe un voltaje en la salida analogica.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Voltage	E	Valor de voltaje a escribir en salida.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Recuerde establecer rango de salida analógica al menos una vez en la librería.

Ejemplo:

```
// Especificar rango de 0-10V en la salida analógica VOUT1.
PioBoard.Cmd.Vout.SetRange(VoutCh.Vout1, VoutVoltRange.Unipolar_10V);

// Escribir 5 Volts en salida Vout1.
Voltage = PioBoard.Cmd.Vout.Vout(VoutCh.Vout1, 5f);
```

Notar que utilizamos “5f” con el sufijo con la letra “f” para explicitar que es una constante del tipo flotante.

float BinaryToVoltage(VoutCh Vout, UInt16 Value): Retorna el voltage de la salida Vout de acuerdo al valor binario pasado.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Value	E	Valor binario a convertir.
Retorno	**Tipo**	**Descripción**
float	S	Voltaje.
Notas		**Descripción**
1		Recuerde establecer rango de salida analógica al menos una vez en la librería.

Ejemplo:

```
// Variable para almacenar voltage.
float Voltage;

// Especificar rango de 0-10V en la salida analógica VOUT1.
PioBoard.Cmd.Vout.SetRange(VoutCh.Vout1, VoutVoltRange.Unipolar_10V);

// Voltage en salida Vout1 correspondiente al valor digital 234
Voltage = PioBoard.Cmd.Vout. BinaryToVoltage(234);
```

UInt16 VoltageToBinary(VoutCh Vout, float Voltage): Convierte un valor binario en el voltaje correspondiente en la salida analogica Vout.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Voltage	E	Voltaje a convertir.
Retorno	**Tipo**	**Descripción**
Value	S	Valor en binario equivalente al voltaje.
Notas		**Descripción**
1		Recuerde establecer rango de salida analógica al menos una vez en la librería.

Ejemplo:

```
// Variable para almacenar valor binario.
UInt16 Value;

// Especificar rango de 0-10V en la salida analógica VOUT1.
PioBoard.Cmd.Vout.SetRange(VoutCh.Vout1, VoutVoltRange.Unipolar_10V);

// Obtener valor binario equivalente a 5V en salida analogica VOUT1.
Value = PioBoard.Cmd.Vout.VoltageToBinary(VoutCh.Vout1, 5f);
```

8.6.4 Métodos para Generador Sinusoidal

SendStat VoutSineEnable(VoutCh Vout): Activa el generador sinusoidal en la salida analogica.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Se puede llamar VoutSineEnable() y activa directamente VOUT1.

Ejemplo:

```
// Activar generador sinusoidal en salida VOUT1.
PioBoard.Cmd.Vout.VoutSineEnable(VoutCh.Vout1);
```

SendStat VoutSineDisable(VoutCh Vout): Desactiva el generador sinusoidal en la salida analógica.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Se puede llamar VoutSineDisable() y desactiva directamente VOUT1.

Ejemplo:

```
// Desactivar generador sinusoidal en salida VOUT1.
PioBoard.Cmd.Vout.VoutSineDisable(VoutCh.Vout1);
```

SendStat VoutSineFrequency(UInt16 Frequency, VoutCh Vout): Establece la frecuencia del generador sinusoidal en la salida analógica.		
Argumentos	**Tipo**	**Descripción**
Vout	E	Selecciona salida analógica.
Frequency	E	Frecuencia en Hz. Máxima frecuencia, ver hojas de datos.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Se puede llamar VoutSineFrequency() y establece directamente VOUT1.

Ejemplo:

```
// Activar generador sinusoidal.
PioBoard.Cmd.Vout.VoutSineEnable(VoutCh.Vout1);

// Establecer frecuencia en 67 Hz.
PioBoard.Cmd.Vout.VoutSineFrequency(67, VoutCh.Vout1);
```

8.6.5 Enumeraciones

enum VoutCh: byte: Enumeracion que representa salidas analogicas.

Campos	Tipo	Valor (dec)	Descripción
Vout1	byte	1	Representa salida VOUT1.
Vout2	byte	2	Representa salida VOUT2.
Vout3	byte	3	Representa salida VOUT3.
Vout4	byte	4	Representa salida VOUT4.
Vout5	byte	5	Representa salida VOUT5.
Vout6	byte	6	Representa salida VOUT6.
Vout7	byte	7	Representa salida VOUT7.
Vout8	byte	8	Representa salida VOUT8.

enum VoutVoltRange: byte: Enumeracion con rangos de voltaje disponibles. Consulte hoja de datos.

Campos	Tipo	Valor (dec)	Descripción
Unipolar_5V	byte	0	Rango de voltaje: 0 a 5V.
Bipolar_5V	byte	1	Rango de voltaje: -5V a 5V.
Unipolar_10V	byte	2	Rango de voltaje: 0 a 10V.
Bipolar_10V	byte	3	Rango de voltaje: -10V a 10V.
Unipolar_3_3V	byte	4	Rango de voltaje: 0 a 3.3V.

8.7 Objeto PioBoard.Cmd.Vin

Contiene comandos para leer y muestrear las entradas analógicas VINx del dispositivo.

Los rangos de tensión pueden configurarse solamente por software o también es necesario establecerlos por hardware (mediante el uso de jumpers). Consulte la hoja de datos del dispositivo.

Es importante destacar que hay dos métodos para leer una entrada analógica:

1. Comando de Lectura (Read): El dispositivo lee una entrada analógica cuando ejecuta el comando.
2. A través de los Samplers: Cuando un sampler es activado, el dispositivo ejecuta un muestreo de las entradas seleccionadas a una frecuencia de muestreo fija, y luego las envía a una dirección IP a través de paquetes UDP-STREAMS.

Para medir señales de muy baja frecuencia (voltajes por ejemplo) se recomienda el primer método. Cuando necesitamos adquirir señales de elevada frecuencia, se recomienda utilizar el segundo método.

Finalmente, cuando necesitamos leer entradas analógicas con señales que varían muy lentamente, por ejemplo sensores de temperatura, tensión continua, entre otros, puede mejorar la precisión de la lectura activando los filtros analógicos, ver más en página 76.

Importante:

- Si la entrada analógica no tiene señal o voltaje conectado (esta al aire), el resultado de una lectura no está definido. Dependiendo del hardware del dispositivo los valores pueden variar.

8.7.1 Métodos de Rangos para Entradas

Permiten establecer los rangos de operación de las entradas analógicas. Es muy importante notificar a la librería sobre el rango empleado para un correcto de la misma.

Algunos dispositivos pueden requerir configuración extra por hardware y/o software para la correcta configuración de los rangos en los canales analógicos. Consulte la hoja de datos del dispositivo.

SendStat SetRange(VinCh Vin, VinVoltRange Range): Establece rango de voltaje en entrada.		
Argumentos	**Tipo**	**Descripción**
Vin	E	Selecciona canal o entrada analógica
Range	E	Rango a establecer. Ver enumeración VinConvStat, pag. 83.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat, pag. 83.
Notas		**Descripción**
1		Algunos dispositivos pueden requerir que además configure el rango de voltaje de salida con un jumper. Consulte hoja de datos del dispositivo.

Ejemplo:

```
// Establecer rango de 0-10V para canal VIN1.
PioBoard.Cmd.Vin.SetRange(VinCh.Vin1, VinVoltRange.Unipolar_10V);
```

SendStat SetRange(VinCh Vin, VinVoltRange Range, float PreAmpGain): Establece rango de voltaje en entrada y ganacia de pre-amplificacion del canal.		
Argumentos	**Tipo**	**Descripción**
Vin	E	Selecciona canal o entrada analógica
Range	E	Rango a establecer. Ver enumeración VinConvStat, pag. 83.
PreAmpGain	E	Ganancia del pre-amplificador a la entrada [2].
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat, pag. 83.
Notas		**Descripción**
1		Algunos dispositivos pueden requerir que además configure el rango de voltaje de salida con un jumper. Consulte hoja de datos del dispositivo.
2		La ganancia de pre-amplificación es un valor interno del dispositivo o valor a medida solicitado por el cliente. Adicionalmente puede utilizarse si el usuario coloca un amplificador antes de la entrada analógica.

Ejemplo:

```
// Establecer rango de 0-10V y  ganacia de 2.5 veces para canal VIN1.
PioBoard.Cmd.Vin.SetRange(VinCh.Vin1, VinVoltRange.Unipolar_10V, 2.5f);
```

Finalmente, los métodos:

- VinVoltRange GetRange(VinCh Vin) : Devuelve el rango configurado para la entrada **Vin**.
- float GetPreAmpGain(VinCh Vin): Devuelve la ganancia configurada para la entrada **Vin**.

8.7.2 Métodos para Lectura de Entradas

SendStat Read(VinCh Vin, out UInt16 Sample, out VinConvStat ConvStat): Lee una entrada analogica. El dispositivo realizará una conversion analogica a digital.

Argumentos	**Tipo**	**Descripción**
Vin	E	Canal o entrada analógica a leer.
Sample	S	Resultado de la lectura digital en la entrada analógica. La resolución depende del modelo (por ej: 10 bits), consulte hoja de datos del dispositivo.
ConvStat	S	Estado de la conversión analógica a digital. Ver enumeración VinConvStat.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Cuando el sampler A esta funcionado, no se pueden leer las siguientes entradas: • STX8091: Todos los canales. • STX8081: Canales VIN1 y VIN2.
2		Cuando el sampler B esta funcionado, no se pueden leer las siguientes entradas: • STX8081: Canales VIN3 y VIN4.
3		El valor de la muestra obtenida puede estar en complemento a 2 para tensiones negativas en algunos dispositivos. Consultar hoja de datos.

Ejemplo:

```
// Establecer rango de 0-10V para canal VIN1 (llamar una sola vez para configurar).
PioBoard.Cmd.Vin.SetRange(VinCh.Vin1, VinVoltRange.Unipolar_10V);

// Variable donde se almacenara la lectura de la entrada analogica.
ushort Sample;

// Variable donde se almacena el estado de la conversion.
VinConvStat ConvStat;

// Leer entrada analogica Vin1 y almacenar resultado en "Sample".
PioBoard.Cmd.Vin.Read(VinCh.Vin1, out Sample, out ConvStat);

// Lectura exitosa ?.
if (ConvStat != VinConvStat.OK)
{
   // No... existio un error al leer la entrada.
   // Por ejemplo: Sampler 1/2 activado.
}
```

Nota: Si existe un error al ejecutar el comando, se puede leer el codigo de estado “SendStat” de la transaccion en el retorno del metodo.

VinStat Read(VinCh Vin, out UInt16 Sample): Lee una entrada analogica. El dispositivo realizará una conversion analogica a digital.		
Argumentos	**Tipo**	**Descripción**
Vin	E	Canal o entrada analógica a leer.
Sample	S	Resultado de la lectura digital en la entrada analógica. La resolución depende del modelo (por ej: 10 bits), consulte hoja de datos del dispositivo.
Retorno	**Tipo**	**Descripción**
VinStat	S	Estado de operación. Ver estructura VinStat.
Notas		**Descripción**
1		Cuando el sampler A esta funcionado, no se pueden leer las siguientes entradas: • STX8091: Todos los canales. • STX8081: Canales VIN1 y VIN2.
2		Cuando el sampler B esta funcionado, no se pueden leer las siguientes entradas: • STX8081: Canales VIN3 y VIN4.
3		El valor de la muestra obtenida puede estar en complemento a 2 para tensiones negativas en algunos dispositivos. Consultar hoja de datos.

Ejemplo:

```
// Establecer rango de 0-10V para canal VIN1 (llamar una sola vez para configurar).
PioBoard.Cmd.Vin.SetRange(VinCh.Vin1, VinVoltRange.Unipolar_10V);

// Variable donde se almacenara la lectura de la entrada analogica.
ushort Sample;

// Variable donde se almacena el estado de operacion.
VinStat Status;

// Leer entrada analogica Vin1 y almacenar resultado en "Sample".
Status = PioBoard.Cmd.Vin.Read(VinCh.Vin1, out Sample);

// Lectura exitosa ?.
if (Status.ConvStat != VinConvStat.OK)
{
    // No... existio un error al leer la entrada.
    // Por ejemplo: Sampler 1/2 activado.
}

// Comando exitoso ?.
if (Status.TransactionStat != SendStat.Success)
{
    // No... error en la transaccion.
}
```

VinStat ReadVoltage(VinCh Vin, out float Voltage): Lee una entrada analogica y retorna el voltage en la entrada. El dispositivo realizará una conversion analogica a digital.

Argumentos	Tipo	Descripción
Vin	E	Canal o entrada analógica a leer.
Voltage	S	Voltaje en la entrada analógica. La resolución depende del modelo (por ej: 10 bits), consulte hoja de datos del dispositivo.
Retorno	**Tipo**	**Descripción**
VinStat	S	Estado de operación. Ver estructura VinStat.
Notas		**Descripción**
1		Cuando el sampler A esta funcionado, no se pueden leer las siguientes entradas: • STX8091: Todos los canales. • STX8081: Canales VIN1 y VIN2.
2		Cuando el sampler B esta funcionado, no se pueden leer las siguientes entradas: • STX8081: Canales VIN3 y VIN4.
3		El voltaje real en la entrada, puede diferir ligeramente.

Ejemplo:

```
// Establecer rango de 0-10V para canal VIN1 (llamar una sola vez para configurar).
PioBoard.Cmd.Vin.SetRange(VinCh.Vin1, VinVoltRange.Unipolar_10V);

// Variable donde se almacenara el voltage de la entrada analogica.
float Voltage;

// Leer entrada analogica Vin1 y almacenar resultado en "Voltage".
PioBoard.Cmd.Vin.ReadVoltage(VinCh.Vin1, out Voltage);

// Tension en entrada Vin1 mayor a 5V ?.
if (Voltage > 5)
{
    // Cerrar RELAY1 (por ejemplo).
    PioBoard.Cmd.Relay.Close(Relays.Relay1);
}
else
{
    // No, Abrir RELAY1.
    PioBoard.Cmd.Relay.Open(Relays.Relay1);
}
```

Ejemplo comprobando errores:

```
// Establecer rango de 0-10V para canal VIN1 (llamar una sola vez para configurar).
PioBoard.Cmd.Vin.SetRange(VinCh.Vin1, VinVoltRange.Unipolar_10V);

// Variable donde se almacenara el voltage de la entrada analogica.
float Voltage;

// Variable donde se almacena el estado de operacion.
VinStat Status;

// Leer entrada analogica Vin1 y almacenar resultado en "Voltage".
Status = PioBoard.Cmd.Vin.ReadVoltage(VinCh.Vin1, out Voltage);

// Lectura exitosa ?.
if (Status.ConvStat != VinConvStat.OK)
{
   // No... existio un error al leer la entrada.
   // Por ejemplo: Sampler A activado.
}

// Tension en entrada Vin1 mayor a 5V ?.
if (Voltage > 5)
{
   // Cerrar RELAY1 (por ejemplo).
   PioBoard.Cmd.Relay.Close(Relays.Relay1);
}
else
{
   // No, Abrir RALAY1.
   PioBoard.Cmd.Relay.Open(Relays.Relay1);
}
```

float BinToVoltage(VinCh Vin, UInt16 BinSample, byte SampleResolution): Retorna el voltage equivalente de una muestra binaria, de acuerdo al canal y la resolucion empleada.

Argumentos	Tipo	Descripción
Vin	E	Canal o entrada analógica donde fue adquirida la muestra.
BinSample	E	Muestra binaria adquirida previamente.
SampleResolution	E	Resolución de la muestra binaria “BinSample”. Por ejemplo: 8,10, 12 bits.
Retorno	**Tipo**	**Descripción**
float	S	Voltaje equivalente de la muestra binaria.
Notas		**Descripción**
-		

Ejemplo:

```
// Establecer rango de 0-10V para canal VIN1 (llamar una sola vez para configurar).
PioBoard.Cmd.Vin.SetRange(VinCh.Vin1, VinVoltRange.Unipolar_10V);

// Variable donde se almacenara la muestra de la entrada analogica.
ushort Sample;

// Leer entrada analogica Vin1 y almacenar resultado en "Sample".
PioBoard.Cmd.Vin.Read(VinCh.Vin1, out Sample);

// Convertir "Sample" a un valor de Voltage equivalente.
// Notar que utilizamos entrada Vin1 y una resolucion de muestra de 10 bits.
float Voltage = PioBoard.Cmd.Vin.BinToVoltage(VinCh.Vin1, Sample, 10);
```

8.7.3 Métodos para Leer Corriente

VinStat ReadCurrent(VinCh Vin, float Resistence, out float Current): Lee corriente en el canal de entrada de tension analogica utilizando un resistor en paralelo.

Argumentos	Tipo	Descripción
Vin	E	Canal o entrada analógica a leer.
Resistence	E	Resistencia del resistor usado para medir corriente (en paralelo a la entrada de tensión).
Current	S	Corriente obtenida por ley de Ohm: **Current = Voltage / Resistence**.
Retorno	**Tipo**	**Descripción**
VinStat	S	Estado de operación. Ver estructura VinStat.
Notas		**Descripción**
1		Cuando el sampler A esta funcionado, no se pueden leer las siguientes entradas: • STX8091: Todos los canales. • STX8081: Canales VIN1 y VIN2.
2		Cuando el sampler B esta funcionado, no se pueden leer las siguientes entradas: • STX8081: Canales VIN3 y VIN4.
3		La corriente real obtenida puede diferir ligeramente.
4		Algunos modelos de dispositivos incorporan un resistor en paralelo a la entrada analógica para leer corriente. Los mismos se pueden seleccionar con un jumper. Consulte hoja de datos del dispositivo.

Ejemplo:

```
// Establecer rango de 0-10V para canal VIN1 (llamar una sola vez para configurar).
PioBoard.Cmd.Vin.SetRange(VinCh.Vin1, VinVoltRange.Unipolar_10V);

// Variable donde se almacenará la corriente que circula por el resistor en paraleo a la entrada.
float Current;

// Leer corriente entrada analogica Vin1 y almacenar resultado en "Current".
// Se utiliza un valor de resistencia para el resistor de 100 Ohms.
PioBoard.Cmd.Vin.ReadVoltage(VinCh.Vin1, 100, out Current);

// Curriente en entrada Vin1 entre 4 mA y 20 mA ?.
if (Current > 0.004f && Current < 0.02f)
{
    // Cerrar RELAY1 (por ejemplo).
    PioBoard.Cmd.Relay.Close(Relays.Relay1);
}
else
{
    // No, Abrir RELAY1.
    PioBoard.Cmd.Relay.Open(Relays.Relay1);
}
```

8.7.4 Filtrado de entradas analógicas

Se proveen funciones para activar filtros digitales en las entradas analógicas. Esto es ideal para eliminar ruidos o pequeñas perturbaciones en señales que varían muy lentamente (como potenciómetros, sensores de temperatura, etc). El tipo de filtro que se utiliza es "**pasa bajos**" mediante promedio de muestras.

SendStat FilterAStart(UInt16 Samples, UInt16 Ts): Activa el filter A. El dispositivo activará un filtro digital para las entradas analógicas. Una vez activado el filtro, la lectura de las entradas analógicas devolverán valores filtrados, ideal para eliminar pequeñas fluctuaciones o ruido en señales que varían muy lentamente.

Argumentos	Tipo	Descripción
Samples	E	Cantidad de nuestras a tomar antes de realizar el filtrado mediante promedio.
Ts	E	Periodo de muestreo en micro-segundos. Ver hoja de datos para máxima frecuencia disponible.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		A mayor periodo de muestreo, menor es la exigencia para el procesador del dispositivo, por lo tanto el desempeño global del dispositivo para otras actividades será mejor. Recuerde detener el filtro si no lo utiliza.
2		Al activar el filtro A, no es posible utilizar el sampler A y viceversa.
3		El filtro digital necesita un tiempo igual a (**TS** * **SAMPLES**) segundos para actualizar su salida. Note que el tiempo descripto anteriormente determina la tasa de refresco de las entradas analógicas afectadas, esto quiere decir que si necesita medir señales que cambian rápidamente, no es aconsejable el uso del filtro digital.
4		En los modelos de dispositivos: • STX8091 el filtro A afecta a todas las entradas analógicas. • STX8081 el filtro A solo afecta a las entradas VIN1 y VIN2.

Ejemplo 1:

```
// Activar filter A para filtrar entradas VIN1 y VIN2. Especificar un periodo de muestreo
// de 2000 uS, es decir una frecuencia de muestreo de Fs = 1/Ts = 500 Hz.
// Tomar 200 muestras para realizar cada filtrado.
PioBoard.Cmd.Vin.FilterAStart(200, 2000);

// Variable donde se almacenara la lectura de la entrada analogica.
ushort Sample;

// Variable donde se almacena el estado de operacion.
VinStat Status;

// Leer entrada analogica Vin1 y almacenar resultado en "Sample".
Status = PioBoard.Cmd.Vin.Read(VinCh.Vin1, out Sample);
```

Notar como el valor de “Sample” ahora es un valor filtrado porque el filtro esta activado. De lo contrario Read() devolveria un valor “imediato” de la entrada analogica.

SendStat FilterBStart(UInt16 Samples, UInt16 Ts): Activa el filter B. El dispositivo activará un filtro digital para las entradas analógicas. Una vez activado el filtro, la lectura de las entradas analógicas devolverán valores filtrados, ideal para eliminar pequeñas fluctuaciones o ruido en señales que varían muy lentamente.

Argumentos	Tipo	Descripción
Samples	E	Cantidad de nuestras a tomar antes de realizar el filtrado mediante promedio.
Ts	E	Periodo de muestreo en micro-segundos. Ver hoja de datos para máxima frecuencia disponible.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		A mayor periodo de muestreo, menor es la exigencia para el procesador del dispositivo, por lo tanto el desempeño global del dispositivo para otras actividades será mejor. Recuerde detener el filtro si no lo utiliza.
2		Al activar el filtro B, no es posible utilizar el sampler B y viceversa.
3		El filtro digital necesita un tiempo igual a (**TS** * **SAMPLES**) segundos para actualizar su salida. Note que el tiempo descripto anteriormente determina la tasa de refresco de las entradas analógicas afectadas, esto quiere decir que si necesita medir señales que cambian rápidamente, no es aconsejable el uso del filtro digital.
4		En los modelos de dispositivos: • STX8091 el filtro B no existe. • STX8081 el filtro A solo afecta a las entradas VIN1 y VIN2.

Ejemplo 1:

```
// Activar filter B para filtrar entradas VIN3 y VIN4. Especificar un periodo de muestreo
// de 2000 uS, es decir una frecuencia de muestreo de Fs = 1/Ts = 500 Hz.
// Tomar 200 muestras para realizar cada filtrado.
PioBoard.Cmd.Vin.FilterBStart(200, 2000);

// Variable donde se almacenara la lectura de la entrada analogica.
ushort Sample;

// Variable donde se almacena el estado de operacion.
VinStat Status;

// Leer entrada analogica Vin3 y almacenar resultado en "Sample".
Status = PioBoard.Cmd.Vin.Read(VinCh.Vin3, out Sample);
```

Notar como el valor de “Sample” ahora es un valor filtrado porque el filtro esta activado. De lo contrario Read() devolveria un valor “imediato” de la entrada analogica.

SendStat FilterAStop(): Detener filter A.		
Argumentos	**Tipo**	**Descripción**
-		
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Detener filter A.
PioBoard.Cmd.Vin.FilterAStop();
```

SendStat FilterBStop(): Detener filter B.		
Argumentos	**Tipo**	**Descripción**
-		
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Detener filter B.
PioBoard.Cmd.Vin.FilterBStop();
```

8.7.5 Métodos para Samplers

Los samplers (muestreadores) permiten configurar el dispositivo para que muestree las entradas analógicas a una frecuencia fija. Esto evita enviar repetidamente comandos para leer una entrada a una frecuencia fija.

Existen dos samplers (sampler A y sampler B). La resolución de los samplers es de 8-bits .

Una vez activado un sampler su operación es autónoma, y muestrea las entradas analógicas hasta que su buffer interno se llena. Una vez lleno el buffer, envía las muestras a una dirección IP (especificada por el usuario) mediante paquetes **UDP-STREAM**.

El usuario lee las muestras recibidas y realiza alguna operación sobre ellas

En pagina 107, sección 9, se explica cómo recibir y manejar paquetes **UDP-STREAM**.

SendStat SamplerAStart(VinSamplerACh Ch, UInt16 Ts): Activa el sampler A. El dispositivo enviará las muestras a través de paquetes UDP-STREAM a una dirección IP especificada.

Argumentos	Tipo	Descripción
Ch	E	Canal o entrada analógica a muestrear (Vin1, Vin2 , etc).
Ts	E	Periodo de muestreo en micro-segundos. Ver hoja de datos para máxima frecuencia disponible.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Todas las muestras obtenidas tienen 8 bits de resolución.
2		A mayor periodo de muestreo, menor es la exigencia para el procesador del dispositivo, por lo tanto el desempeño global del dispositivo para otras actividades será mejor. Recuerde detener el sampler si no lo utiliza en periodos prolongados.
3		En los modelos de dispositivo: • STX8091 el sampler A se puede utilizar en todas las entradas analógicas. • STX8081 el sampler A se puede utilizar con VIN1 y VIN2.
4		Cuando este sampler es utilizado, dependiendo del modelo de PLC, no se pueden utilizar comandos de lectura en algunos canales analógicos. En los modelos de PLC: • STX8091 el sampler A afecta a todas las entradas analógicas. • STX8081 el sampler A solo afecta a las entradas VIN1 y VIN2.
5		El valor de la muestra obtenida puede estar en complemento a 2 para tensiones negativas en algunos dispositivos. Consultar hoja de datos.

Ejemplo 1:

```
// Especificar dirección IP de la maquina que recibirá las muestras
// obtenidas por el sampler.
PioBoard.Cmd.BoardConfig.SetUdpStreamHostIp(192, 168, 1, 10);

// Activar sampler A, muestrear canal Vin1 con un Ts = 250 uS,
// equivalente a una frecuencia de muestreo de 4000 Hz.
// Las muestreas se enviaran a través de paquetes UDP-STREAM a la dirección
// IP especificada anteriormente.
PioBoard.Cmd.Vin.SamplerAStart(VinSamplerACh.Vin1, 250);
```

Ejemplo 2:

```
// Para adquirir los canales Vin1 y Vin2 al mismo tiempo, utilice:
PioBoard.Cmd.Vin.SamplerAStart(VinSamplerACh.Vin1 | VinSamplerACh.Vin2, 250);
```

SendStat SamplerAStop(): Detener sampler A.		
Argumentos	**Tipo**	**Descripción**
-		
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Detener sampler A.
PioBoard.Cmd.Vin.SamplerAStop();
```

SendStat SamplerBStart(VinSamplerBCh Ch, UInt16 Ts): Activa el sampler A. El dispositivo enviará las muestras a través de paquetes UDP-STREAM a una dirección IP especificada.

Argumentos	Tipo	Descripción
Ch	E	Canal o entrada analógica a muestrear.
Ts	E	Periodo de muestreo en micro-segundos. Ver hoja de datos para máxima frecuencia disponible.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Todas las muestras obtenidas tienen 8 bits de resolución.
2		A mayor periodo de muestreo, menor es la exigencia para el procesador del dispositivo, por lo tanto el desempeño global del dispositivo para otras actividades será mejor. Recuerde detener el sampler si no lo utiliza en periodos prolongados.
3		En los modelos de dispositivo: • STX8091 el sampler B no existe. • STX8081 el sampler A se puede utilizar con VIN3 y VIN4.
4		Cuando este sampler es utilizado, dependiendo del modelo de PLC, no se pueden utilizar comandos de lectura en algunos canales analógicos. En los modelos de PLC: • STX8091 el sampler B no existe. • STX8081 el sampler B solo afecta a las entradas VIN3 y VIN4.
5		El valor de la muestra obtenida puede estar en complemento a 2 para tensiones negativas en algunos dispositivos. Consultar hoja de datos.

Ejemplo 1:

```
// Especificar dirección IP de la maquina que recibirá las muestras
// obtenidas por el sampler.
PioBoard.Cmd.BoardConfig.SetUdpStreamHostIp(192, 168, 1, 10);

// Activar sampler A, muestrear canal Vin3 con un Ts = 250 uS,
// equivalente a una frecuencia de muestreo de 4000 Hz.
// Las muestreas se enviaran a través de paquetes UDP-STREAM a la dirección
// IP especificada anteriormente.
PioBoard.Cmd.Vin.SamplerBStart(VinSamplerBCh.Vin3, 250);
```

Ejemplo 2:

```
// Para adquirir los canales Vin3 y Vin4 al mismo tiempo, utilice:
PioBoard.Cmd.Vin.SamplerBStart(VinSamplerBCh.Vin3 | VinSamplerBCh.Vin4, 250);
```

SendStat SamplerBStop(): Detener sampler B.		
Argumentos	**Tipo**	**Descripción**
-		
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Detener sampler B.
PioBoard.Cmd.Vin.SamplerBStop();
```

8.7.6 Enumeraciones

enum VinCh : byte: Enumeración que representa las entradas analógicas disponibles.			
Campos	**Tipo**	**Valor (dec)**	**Descripción**
Vin1	byte	1	Entrada analógica 1.
Vin2	byte	2	Entrada analógica 2.
Vin3	byte	3	Entrada analógica 3.
Vin4	byte	4	Entrada analógica 4.
Vin5	byte	5	Entrada analógica 5.
Vin6	byte	6	Entrada analógica 6.
Vin7	byte	7	Entrada analógica 7.
Vin8	byte	8	Entrada analógica 8.

enum VinConvStat : byte: Enumeración que representa el estado de una conversión analógica a digital.			
Campos	**Tipo**	**Valor (dec)**	**Descripción**
OK	byte	0	Conversión exitosa.
Error	byte	1	Conversión errónea.

enum VinSamplerACh : byte: Enumeración que representa las entradas analógicas para utilizar en el sampler A.

Campos	Tipo	Valor (bin)	Descripción
Vin1	byte	0000 0001	Entrada analógica 1.
Vin2	byte	0000 0010	Entrada analógica 2.
Vin1and2	byte	0000 0011	Entradas analógica 1 y 2.
Vin3	byte	0000 0100	Entrada analógica 3.
Vin4	byte	0000 1000	Entrada analógica 4.
Vin5	byte	0001 0000	Entrada analógica 5.
Vin6	byte	0010 0000	Entrada analógica 6.
Vin7	byte	0100 0000	Entrada analógica 7.
Vin8	byte	1000 0000	Entrada analógica 8.

enum VinSamplerBCh : byte: Enumeración que representa las entradas analógicas disponibles para utilizar en el sampler 3/4.

Campos	Tipo	Valor (bin)	Descripción
Vin3	byte	01	Entrada analógica 3.
Vin4	byte	10	Entrada analógica 4.
Vin3and4	byte	11	Entradas analógica 3 y 4.

enum VinVoltRange : byte: Enumeración que representa las constantes para establecer rangos de tensión en entradas analógicas. Algunos modelos de dispositivos o entradas, pueden aceptar solo algunos de estos rangos.

Campos	Tipo	Valor (dec)	Descripción
Unipolar_5V	byte	0	Rango 0V a +5V.
Bipolar_5V	byte	1	Rango -5V a +5V. Bipolar.
Unipolar_10V	byte	2	Rango 0V a +10V.
Bipolar_10V	byte	3	Rango -10V a +10V. Bipolar.
Unipolar_3_3V	byte	4	Rango 0V a +3.3V.

8.7.7 Estructuras

struct VinStat: Estructura con el estado retornado por algunos métodos para leer entradas analógicas.			
Campos	**Tipo**	**Valor**	**Descripción**
ConvStat	VinConvStat	-	Estado de la ultima conversión ejecutada. Ver enumeración VinConvStat.
TransactionStat	SendStat	-	Estado de la ultima transacción al ejecutar un comando. Ver enumeración SendStat.

8.8 Objeto PioBoard.Cmd.Count

Contiene comandos para configurar los contadores COUNT1 y COUNT2. Las entradas asociadas con estos contadores están compartidas con las entradas digitales o discretas (por ejemplo DIN7 y DIN8 respectivamente), consulte la hoja de datos del dispositivo.

Los contadores permiten contar eventos, medir tiempo entre eventos (frecuencia) y avisar a través de paquetes UDP-STREAM si los contadores han llegado a cierto número de cuentas preestablecido por el usuario.

En la hoja de datos del dispositivo, podrá encontrar la definición de un evento.

8.8.1 Métodos

8.8.2 Métodos para Contador 1

SendStat Count1Enable(CountEdge Edge): Inicializa y activa el contador 1 (COUNT1) para la cuenta de eventos.		
Argumentos	**Tipo**	**Descripción**
Edge	E	Selecciona flanco del evento a contar. Ver enumeración CountEdge.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo 1:

```
// Activar cuenta de eventos por flanco ascendente (Rising Edge).
PioBoard.Cmd.Count.Count1Enable(CountEdge.Rising);
```

Ejemplo 2:

```
// Activar cuenta de eventos por flanco descendente (Falling Edge).
PioBoard.Cmd.Count.Count1Enable(CountEdge.Falling);
```

Ejemplo 3:

```
// Activar cuenta de eventos por ambos flancos ascendente/descendente (Both Edges).
PioBoard.Cmd.Count.Count1Enable(CountEdge.Both);
```

SendStat Count1Disable(): Desactiva el contador 1 (COUNT1). Detiene todas las operaciones en progreso.

Argumentos	Tipo	Descripción
-		
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Desactivar contador.
PioBoard.Cmd.Count.Count1Disable();
```

SendStat Count1ChangeEdge(CountEdge Edge): Cambia el flanco del contador 1 (COUNT1).

Argumentos	Tipo	Descripción
Edge	E	Selecciona flanco del evento a contar. Ver enumeración CountEdge.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Activar cuenta de eventos por flanco ascendente (Rising Edge).
PioBoard.Cmd.Count.Count1Enable(CountEdge.Rising);

// ... CODIGO DEL USUARIO ...

// Ahora le decimos al contador que cuente eventos  por
// ambos flancos (Both Edges).
PioBoard.Cmd.Count.Count1ChangeEdge(CountEdge.Both);
```

SendStat Count1Reset(): Resetea (limpia) todos los registros del contador 1 (COUNT1). Valores match y registro de tiempos de eventos son reseteados.		
Argumentos	**Tipo**	**Descripción**
-		
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Resetear contador.
PioBoard.Cmd.Count.Count1Reset();
```

SendStat Count1GetEventCnt(out UInt32 Count): Obtener valor del contador de eventos del contador 1 (COUNT1).		
Argumentos	**Tipo**	**Descripción**
Count	S	Numero de eventos contados por el contador. Valor máximo de 32-bits.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);

// Activar cuenta de eventos por flanco ascendente (Rising Edge).
PioBoard.Cmd.Count.Count1Enable(CountEdge.Rising);

// Variable que almacena numero de eventos contados.
UInt32 Count;

// Obtener el numero de eventos contados por el contador
// y almacenarlo en variable "Count".
PioBoard.Cmd.Count.Count1GetEventCnt(out Count);

// Si el numero de cuentas es mayor a 1000, activar RELAY1.
if (Count > 1000)
{
   PioBoard.Cmd.Relay.Close(Relays.Relay1);
}
```

SendStat Count1GetEventPeriod(out UInt32 Period): Obtiene el tiempo en micro-segundos medido entre dos eventos contados por el contador 1 (COUNT1).		
Argumentos	**Tipo**	**Descripción**
Period	S	Tiempo entre dos eventos sucesivos contados por el contador [1].
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Si el valor de la variable es cero, puede deberse a: • Tiempo entre eventos es menor a 1 uS. • No se han contados eventos todavía. • Error al ejecutar el comando (verificar retorno SendStat).

Ejemplo:

```
// Activar cuenta de eventos por flanco ascendente (Rising Edge).
PioBoard.Cmd.Count.Count1Enable(CountEdge.Rising);

// Variable que almacena el tiempo entre dos eventos contados.
UInt32 PeriodUS;

// Obtener el tiempo entre dos eventos contados por el contador
// y almacenarlo en variable "Period".
PioBoard.Cmd.Count.Count1GetEventPeriod(out PeriodUS);

// Verificar si el tiempo es válido.
if (PeriodUS == 0)
{
   // Error, no se contaron eventos.
   return;
}

// Calcular frecuencia (en caso de tener a la entrada
// una onda cuadrada). Como contamos eventos por flanco ascendente,
// su separación, nos da el periodo, y la inversa la frecuencia.

// Convertimos el periodo a segundos (estaba en micro-segundos).
float Seconds = PeriodUS * 0.000001f;

// Obtenemos frecuencia.
UInt32 Frequency = (UInt32) (1 / Seconds);  // Reemplazar por float para mayor resolución.

// Obtener RPM.
UInt32 RPM = Frequency * 60;  // Reemplazar tipo de variable por float para mayor resolución.
```

Nota: Si contamos por ambos flancos (CountEdge.Both) y tenemos una onda cuadrada como entrada en el contador, obtendríamos Ancho de pulso. Si el DutyCycle de la onda es distinto al 50%, el valor medido alternaría entre el tiempo del pulso en alto y el tiempo del pulso en bajo.

SendStat Count1SetMatch(Int32 MatchValue): Especifica un “valor match” para el contador 1 (COUNT1). Cuando la cuenta de eventos alcance el “valor match”, el dispositivo informará la ocurrencia a través de un paquete UDP-STREAM. El usuario podrá leer ese valor y tomar una decisión.		
Argumentos	**Tipo**	**Descripción**
MatchValue	E	Especifica el valor “match”. Cuando el contador alcance este valor, informara a una dirección IP a través de paquetes UDP-STREAM la ocurrencia. Luego del evento “match”, el contador de eventos es reseteado a cero automáticamente.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Más adelante en el documento, se explica cómo recibir paquetes UDP-STREEAM.

Ejemplo:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);

// Especificar dirección IP de la maquina que recibirá el evento “match”.
PioBoard.Cmd.BoardConfig.SetUdpStreamHostIp(192, 168, 1, 10);

// Activar cuenta de eventos por flanco ascendente (Rising Edge).
PioBoard.Cmd.Count.Count1Enable(CountEdge.Rising);

// Cuando el contador de eventos alcance el numero 3000,
// avisara a una dirección IP a través de un paquete
// UDP-STREAM la ocurrencia de un evento "match".
PioBoard.Cmd.Count.Count1SetMatch(3000);

// Resto del código del usuario….
```

8.8.3 Métodos para Contador 2

SendStat Count2Enable(CountEdge Edge): Inicializa y activa el contador 2 (COUNT2) para la cuenta de eventos.		
Argumentos	**Tipo**	**Descripción**
Edge	E	Selecciona flanco del evento a contar. Ver enumeración CountEdge.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo 2:

```
// Activar cuenta de eventos por flanco ascendente (Rising Edge).
PioBoard.Cmd.Count.Count2Enable(CountEdge.Rising);
```

Ejemplo 2:

```
// Activar cuenta de eventos por flanco descendente (Falling Edge).
PioBoard.Cmd.Count.Count2Enable(CountEdge.Falling);
```

Ejemplo 3:

```
// Activar cuenta de eventos por ambos flancos ascendente/descendente (Both Edges).
PioBoard.Cmd.Count.Count2Enable(CountEdge.Both);
```

SendStat Count2Disable(): Desactiva el contador 2 (COUNT2). Detiene todas las operaciones en progreso.		
Argumentos	**Tipo**	**Descripción**
-		
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Desactivar contador.
PioBoard.Cmd.Count.Count2Disable();
```

SendStat Count2ChangeEdge(CountEdge Edge): Cambia el flanco del contador 2 (COUNT2).		
Argumentos	**Tipo**	**Descripción**
Edge	E	Selecciona flanco del evento a contar. Ver enumeración CountEdge.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Activar cuenta de eventos por flanco ascendente (Rising Edge).
PioBoard.Cmd.Count.Count2Enable(CountEdge.Rising);

// ... CODIGO DEL USUARIO ...

// Ahora le decimos al contador que cuente eventos  por
// ambos flancos (Both Edges).
PioBoard.Cmd.Count.Count2ChangeEdge(CountEdge.Both);
```

SendStat Count2Reset(): Resetea (limpia) todos los registros del contador 2 (COUNT2). Valores match y refistro de tiempos de eventos son reseteados.		
Argumentos	**Tipo**	**Descripción**
-		
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Resetear contador.
PioBoard.Cmd.Count.Count2Reset();
```

SendStat Count2GetEventCnt(out UInt32 Count): Obtener valor del contador de eventos del contador 2 (COUNT2).		
Argumentos	**Tipo**	**Descripción**
Count	S	Numero de eventos contados por el contador. Valor máximo de 32-bits.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);

// Activar cuenta de eventos por flanco ascendente (Rising Edge).
PioBoard.Cmd.Count.Count2Enable(CountEdge.Rising);

// Variable que almacena numero de eventos contados.
UInt32 Count;

// Obtener el numero de eventos contados por el contador
// y almacenarlo en variable "Count".
PioBoard.Cmd.Count.Count2GetEventCnt(out Count);

// Si el numero de cuentas es mayor a 1000, activar RELAY1.
if (Count > 1000)
{
   PioBoard.Cmd.Relay.Close(Relays.Relay1);
}
```

SendStat Count2GetEventPeriod(out UInt32 Period): Obtiene el tiempo en micro-segundos medido entre dos eventos contados por el contador 2 (COUNT2).

Argumentos	Tipo	Descripción
Period	S	Tiempo entre dos eventos sucesivos contados por el contador [1].
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Si el valor de la variable es cero, puede deberse a: • Tiempo entre eventos es menor a 1 uS. • No se han contados eventos todavía. • Error al ejecutar el comando (verificar retorno SendStat).

Ejemplo:

```
// Activar cuenta de eventos por flanco ascendente (Rising Edge).
PioBoard.Cmd.Count.Count2Enable(CountEdge.Rising);

// Variable que almacena el tiempo entre dos eventos contados.
UInt32 PeriodUS;

// Obtener el tiempo entre dos eventos contados por el contador
// y almacenarlo en variable "Period".
PioBoard.Cmd.Count.Count2GetEventPeriod(out PeriodUS);

// Verificar si el tiempo es válido.
if (PeriodUS == 0)
{
   // Error, no se contaron eventos.
   return;
}

// Calcular frecuencia (en caso de tener a la entrada
// una onda cuadrada). Como contamos eventos por flanco ascendente,
// su separación, nos da el periodo, y la inversa la frecuencia.

// Convertimos el periodo a segundos (estaba en micro-segundos).
float Seconds = PeriodUS * 0.000001f;

// Obtenemos frecuencia.
UInt32 Frequency = (UInt32) (1 / Seconds);  // Reemplazar por float para mayor resolución.

// Obtener RPM.
UInt32 RPM = Frequency * 60;   // Reemplazar tipo de variable por float para mayor resolución.
```

Nota: Si contamos por ambos flancos (CountEdge.Both) y tenemos una onda cuadrada como entrada en el contador, obtendríamos Ancho de pulso. Si el DutyCycle de la onda es distinto al 50%, el valor medido alternaría entre el tiempo del pulso en alto y el tiempo del pulso en bajo.

SendStat Count2SetMatch(Int32 MatchValue): Especifica un "valor match" para el contador 2 (COUNT2). Cuando la cuenta de eventos alcance el "valor match", el dispositivo informará la ocurrencia a través de un paquete UDP-STREAM. El usuario podrá leer ese valor y tomar una decisión.

Argumentos	**Tipo**	**Descripción**
MatchValue	E	Especifica el valor "match". Cuando el contador alcance este valor, informara a una dirección IP a través de paquetes UDP-STREAM la ocurrencia. Luego del evento "match", el contador de eventos es reseteado a cero automáticamente.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Más adelante en el documento, se explica cómo recibir paquetes UDP-STREEAM.

Ejemplo:

```
// Se crea un objeto PioBoard del tipo Stx8081.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);

// Especificar dirección IP de la maquina que recibirá el evento "match".
PioBoard.Cmd.BoardConfig.SetUdpStreamHostIp(192, 168, 1, 10);

// Activar cuenta de eventos por flanco ascendente (Rising Edge).
PioBoard.Cmd.Count.Count2Enable(CountEdge.Rising);

// Cuando el contador de eventos alcance el numero 3000,
// avisara a una dirección IP a través de un paquete
// UDP-STREAM la ocurrencia de un evento "match".
PioBoard.Cmd.Count.Count2SetMatch(3000);

// Resto del código del usuario….
```

8.8.4 Enumeraciones

enum CountEdge : byte: Enumeración que representa los flancos validos para contar eventos con los contadores.

Campos	Tipo	Valor (dec)	Descripción
None	byte	0	Ningún flanco de cuenta. No utilizar.
Falling	byte	1	Flanco descendente para cuenta de eventos.
Rising	byte	2	Flanco ascendente para cuenta de eventos.
Both	byte	3	Ambos flancos (descendente/ascendente) para contar eventos.

8.9 Objeto PioBoard.Cmd.Udp

Contiene comandos y métodos para enviar y recibir datos UDP a programas en lenguaje PAWN / LADDER que estén ejecutándose en el dispositivo en modo PLC.

Este objeto es descripto adecuadamente en el "Manual de Usuario Modo PLC" del dispositivo. Allí se explica cómo enviar y recibir datos UDP desde programas que se ejecutan en el dispositivo.

También es recomendado leer la nota de aplicación **STX-AN001** disponible en nuestra página Web.

8.10 Objeto PioBoard.Cmd.Board

Comandos generales para controlar el dispositivo STX80XX.

8.10.1 Métodos

SendStat Reset(): Aplica un RESET al microprocesador del dispositivo.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Tenga en cuenta que este comando no verifica el dispositivo fue reseteado correctamente, debido a que el dispositivo no responde a este comando. Solo es reseteado.

Ejemplo:

Resetear dispositivo:

```
PioBoard.Cmd.Board.Reset();
```

Luego de este comando el dispositivo se inicializará nuevamente. Recuerde que la inicialización toma un pequeño tiempo, que deberá tener en cuenta antes de enviar el próximo comando.

SendStat Ping(): Solo envia un comando para comprobar la conexión. No tiene efecto alguno.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
SendStat CmdStat = SendStat.Success;

CmdStat = PioBoard.Cmd.Board.Ping();

if (CmdStat != SendStat.Success)
{
    MessageBox.Show(CmdStat.ToString(), "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

8.11 Objeto PioBoard.Cmd.BoardConfig

Comandos para configurar el dispositivo.

La configuración del dispositivo puede ser realizada de forma simple a través del programa ***BoardConfig*** disponible en el paquete STX80XX-SDK suministrado.

8.11.1 Métodos

SendStat SetMAC(byte A0, byte A1, byte A2, byte A3, byte A4, byte A5): Cambia la direccion MAC asignada al dispositivo en una red Ethernet.		
Argumentos	**Tipo**	**Descripción**
A0	E	Dirección MAC, byte 0.
A1	E	Dirección MAC, byte 1.
A2	E	Dirección MAC, byte 2.
A3	E	Dirección MAC, byte 3.
A4	E	Dirección MAC, byte 4.
A5	E	Dirección MAC, byte 5.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		El nuevo valor de MAC será utilizado luego del próximo RESET del dispositivo.

Ejemplo:

Cambiar la dirección MAC ethernet del dispositivo a 34:34:44:63:1:55 (decimal):

PioBoard.Cmd.BoardConfig.SetMAC(34, 34, 55, 63, 1, 55);

Nota: Puede usar valores hexadecimales con el prefijo: “0x”
Por ejemplo: A0 –> 0xA0

SendStat SetIP(byte A3, byte A2, byte A1, byte A0): Cambia la direccion IP asignado al dispositivo en una red TCP/IP.

Argumentos	Tipo	Descripción
A3	E	Dirección IP, primer octeto.
A2	E	Dirección IP, segundo octeto.
A1	E	Dirección IP, tercer octeto.
A0	E	Dirección IP, cuarto octeto.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		El nuevo valor de IP será utilizado luego del próximo RESET del dispositivo.
2		Verifique que la NetMask del dispositivo sea el correcto al especificar una nueva IP.

Ejemplo:

Cambiar la dirección IP actual del dispositivo a la dirección “192.168.1.15”:

PioBoard.Cmd.BoardConfig.SetIP(192, 168, 1, 15);

SendStat SetNetmask(byte A3, byte A2, byte A1, byte A0): Cambia la mascara de red “NetMask” asignada al dispositivo en una red TCP/IP.

Argumentos	Tipo	Descripción
A3	E	Dirección NetMask, primer octeto.
A2	E	Dirección NetMask, segundo octeto.
A1	E	Dirección NetMask, tercer octeto.
A0	E	Dirección NetMask, cuarto octeto.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		El nuevo valor de IP será utilizado luego del próximo RESET del dispositivo.

Ejemplo:

Si el dispositivo tiene una dirección IP de clase C (ej: 192.168.1.15), la dirección NetMask que le corresponde es la siguiente:

PioBoard.Cmd.BoardConfig.SetNetmask(255, 255, 255, 0);

SendStat SetPassword(UInt32 Password): Cambia el password del dispositivo. Para poder ejecutar comandos en el dispositivo, usted debe recordar el password establecido. Por fabrica el valor por defecto es 0.		
Argumentos	**Tipo**	**Descripción**
Password	E	Nuevo password del dispositivo. Valor entero de 32-bits.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Recuerde especificar el nuevo password en la librería con el método PioBoard.Globals.SetCmdServerPassword(Password) la proxima vez que instancie la librería.

Ejemplo:

Cambiar el password de fábrica, por el password “2349092”.

PioBoard.Cmd.BoardConfig.SetPassword(2349092);

La proxima vez que utilizemos la librería deberemos especificar este nuevo password para enviar comandos al dispositivo.

SendStat SetUdpStreamHostIp(byte A3, byte A2, byte A1, byte A0): Especifica la direccion IP del Host o Computadora donde el dispositivo STX80XX enviará los paquetes UDP-STREAM.		
Argumentos	**Tipo**	**Descripción**
A3	E	Dirección IP, primer octeto.
A2	E	Dirección IP, segundo octeto.
A1	E	Dirección IP, tercer octeto.
A0	E	Dirección IP, cuarto octeto.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Este método especifica que computadora recibirá paquetes UDP-STREAM. Un paquete UDP-STREAM es un flujo de datos continuo que el dispositivo STX80XX envía asincrónicamente a una dirección IP. El contenido del paquete puede ser: Datos de una señal analógica, un cambio en alguna entrada discreta, counter match, etc.

Ejemplo:

Supongamos que activamos el sampler A para muestrear la entrada analógica VIN1, el dispositivo enviará las muestras a la dirección IP siguiente (que puede ser nuestra computadora):

PioBoard.Cmd.BoardConfig.SetUdpStreamHostIp(192, 168, 1, 11);

SendStat SetRtcTime(int Hour, int Min, int Sec, int Day, int Month, int Year): Establece hora y fecha del reloj en tiempo real (RTC) del dispositivo.		
Argumentos	**Tipo**	**Descripción**
Hour	E	Hora (0-23).
Min	E	Minuto (0-59).
Sec	E	Segundos (0-59).
Day	E	Día (1-31).
Month	E	Mes (1-12).
Year	E	Año (0-4095).
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		No hace falta reiniciar el dispositivo.

Ejemplo:

Establecer fecha y hora del dispositivo, 3 de Mayo del 2015, 15:33:33 horas.

PioBoard.Cmd.BoardConfig.SetRtcTime(15, 33, 33, 3, 5, 2015);

SendStat LcdEnable(bool Op): Activa o Desativa el uso del display LCD en el dispositivo.		
Argumentos	**Tipo**	**Descripción**
Op	E	Operación. Con valor “True” activamos el LCD y con “False” lo desactivamos.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		No todos los dispositivos cuentan con display LCD incorporado. Consulte hoja de datos.
2		Desactivar el LCD disminuye el tiempo de inicialización del dispositivo, especialmente si no está conectado.

SendStat LcdWellcome(bool Op): Activa o Desativa el mensaje de bienvenida al inicializarce el dispositivo.

Argumentos	Tipo	Descripción
Op	E	Operación. Con valor "True" activamos y con "False" lo desactivamos.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		No todos los dispositivos cuentan con display LCD incorporado. Consulte hoja de datos.

SendStat FeedWatchdog (UInt32 Seconds): Activa y alimenta el watchdog del dispositivo. Una vez activado el watchdog no es posible desactivarlo. Por lo tanto deberia enviar este comando antes del tiempo de expiracion del watchdog para evitar que el dispositivo se reseteé por watchdog.		
Argumentos	**Tipo**	**Descripción**
Seconds	E	Tiempo en segundos que deben transcurrir antes que el watchdog del dispositivo provoque un reset. Valor entero de 32-bits.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
1		Este comando puede utilizarse en modo DAQ y PLC.
2		Recuerde en su programa enviar este comando en intervalos menores al establecido por la entrada "**Seconds**" para evitar que el dispositivo se reseteé.
3		En modo DAQ, el dispositivo ya utiliza el watchdog y lo alimenta automáticamente, pero al enviar este comando, la alimentación pasa a ser manual por parte del usuario (enviando este comando periódicamente).
4		En modo PLC puede utilizar este comando con el mismo objetivo.

Ejemplo:

```
//
// Activar watchdog y alimentar timeout por 30 segundos.
//

PioBoard.Cmd.BoardConfig.FeedWatchdog(30);
```

Según el ejemplo, el watchdog se establecerá para que provoque un reset en el dispositivo a los 30 segundos. Por lo tanto para evitar el reset, debe enviar nuevamente este comando antes de los 30 segundos para alimentar nuevamente al watchdog.

Puede emplear un intervalo de alimentacion un 30% menor al establecido por el watchdog (para tener tiempo extra para posibles fallos de comunicación y reenviar el comando).

Tip: Puede emplear un timer en su aplicación de C# y en cada tick() del timer, enviar el comando FeedWatchdog().

8.12 Objeto PioBoard.Cmd.BoardInfo

Provee métodos para extraer información del dispositivo STX80XX.

8.12.1 Métodos

SendStat GetInfo(out BoardInformation Info): Extrae informacion del dispositivo.		
Argumentos	**Tipo**	**Descripción**
Info	S	Retorno de la información extraída en una estructura del tipo BoardInformation.
Retorno	**Tipo**	**Descripción**
SendStat	S	Estado de transacción. Ver enumeración SendStat.
Notas		**Descripción**
-		

Ejemplo:

```
// Estructura que contendra la informacion del dispositivo.
BoardInformation BoardInfo = new BoardInformation();

// Extraer informacion del dispositivo y almacenarla en BoardInfo.
PioBoard.Cmd.BoardInfo.GetInfo(out BoardInfo);

// Escribir en caja de texto la version de firmware del dispositivo.
textBox1.Text = BoardInfo.FirmwareVersion.ToString();
```

8.12.2 Estructuras

struct BoardInformation: Estructura que almacena informacion del dispositivo STX80XX.		
Campos	**Tipo**	**Descripción**
FirmwareVersion	UInt16	Version del firmware. Numero positivo.
FirmwareDate	UInt32	Fecha de creación del firmware. Formato YYYYMMDD, ej 20090828.
FirmwareName	string	Nombre del firmware instalado.
Manufacturer	string	Nombre del fabricante del dispositivo.
ContactInfo	string	Información de contacto (mail o web).
CEO	string	Actual CEO del fabricante.
IP	string	Dirección IP del dispositivo.
Netmask	string	Mascara de red del dispositivo.

struct BoardInformation (continuación)

Campos	Tipo	Descripción
MAC	byte[]	Array de 6 bytes con la dirección MAC utilizada por el dispositivo.
MACFactory	byte[]	Array de 6 bytes con la dirección MAC especificada por fabrica.
Password	UInt32	Password del dispositivo.
SerialNumber	UInt32	Serial Number (S/N) del dispositivo.
PartNumber	string	Part Number (P/N) del dispositivo.
BooloaderStart	bool	Refleja si el bootloader será activado en el próximo RESET.
LcdEnabled	bool	Refleja si el LCD esta activado (true) o desactivado (false).
LcdWellcome	bool	Refleja si el mensaje de bienvenida del LCD esta activado (true) o desactivado (false).

9 Objetos para Paquetes UDP-STREAM

9.1 Introducción

Los paquetes UDP-STREAM son datos enviados por el dispositivo STX80XX cuando algún evento, previamente activado por el usuario ocurre.

Ejemplo de aplicaciones que requieran el uso de paquetes UDP-STREAM son:

1. El usuario utiliza los contadores del dispositivo para contar cuantas botellas de gaseosa son llenadas en una fábrica envasadora. Si desea parar la producción cuando el numero de botellas sea 100000, tiene dos alternativas:

    a. Enviar constantemente un comando y ver el valor del contador.
    b. Establecer el valor 100000 en el valor "match" del contador y esperar a que el dispositivo avise mediante un paquete UDP-STREAM la ocurrencia del evento "match".

    La alternativa (a) es simple, pero requiere constantemente enviar un comando al dispositivo.

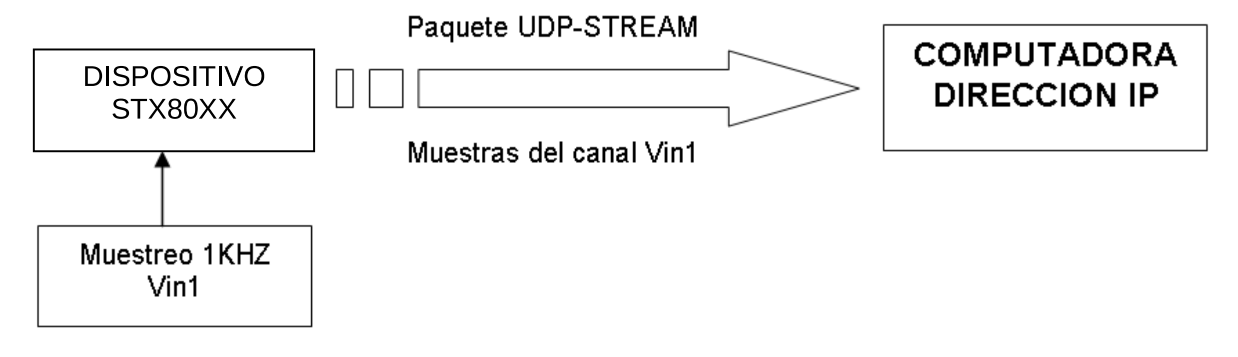
    La alternativa (b) permite al usuario solo esperar que el número del contador sea igual al valor "match" establecido, y cuando ocurra, el dispositivo avisará mediante un paquete UDP-STREAM.

2. Se desea muestrear una señal continuamente a una frecuencia de 1 KHz en la entrada analógica Vin1. El usuario puede recurrir a dos alternativas:

    a. Utilizar el comando Read() para leer continuamente la señal en la entrada Vin1.
    b. Activar el sampler A con una frecuencia de muestreo igual a 1KHz sobre la entrada Vin1, y esperar que el dispositivo envíe las muestras a través de paquetes UDP-STREAM.

    La alternativa (a) no es posible, debido a que no es confiable enviar comandos a una tasa de 1KHz en una red Ethernet utilizando Windows. Esto se debe a que puede existir congestión en la red o el sistema operativo puede generar retardos que provoquen que la frecuencia de muestreo sea menor a 1 KHz.

    La alternativa (b) es la más confiable, debido a que el dispositivo realiza el muestreo a 1KHz sin la intervención de una computadora. Almacena las muestras en un pequeño buffer y luego las envía a una dirección IP (computadora) a través de paquetes UDP-STREAM.
    El usuario recibe las muestras y las almacena, para luego graficarlas, procesarlas o guardarlas en un archivo.



© 2015 Slicetex Electronics

9.2 Paquete UDP-STREAM

Un paquete UDP-STREAM es un paquete de datos, que el dispositivo STX80XX envía asincrónicamente cuando ocurre un evento previamente activado y configurado por el usuario.

El paquete UDP-STREAM se transmite utilizando el protocolo UDP, al puerto 4952.

Tiene la siguiente estructura:

ID	DataSize	Data Bytes

Donde:

- **ID** : Identificador de paquete.
- **DataSize** : Numero de bytes de datos.
- **Data Bytes** : Bytes de datos del paquete. Tamaño variable.

El usuario, para identificar el contenido de los datos que el paquete transporta, debe leer el campo "**ID**" del paquete, el cual a través de un código, nos informa la naturaleza de los datos. El campo "**DataSize**" nos indicará el número de bytes de datos contenido en el campo "**Data Bytes**".

Los identificadores de paquetes "**ID**" posibles para un paquete UDP-STREAM, se describen en la enumeración UdpStreamPacketIDs:

enum UdpStreamPacketIDs : byte: Enumeración que contiene los posibles "ID" de los paquetes UDP-STREAM, y que permiten identificar el tipo de dato transportado.

Campos	**Tipo**	**Valor (dec)**	**Descripción**
None	byte	0	Datos no definidos en el paquete.
VIN1	byte	1	El paquete contiene muestras del canal VIN1.
VIN2	byte	2	El paquete contiene muestras del canal VIN2.
VIN3	byte	3	El paquete contiene muestras del canal VIN3.
VIN4	byte	4	El paquete contiene muestras del canal VIN4.
VIN1AND2	byte	5	El paquete contiene muestras del canal VIN1 en la primera mitad y muestras del canal VIN2 en la segunda mitad.
VIN3AND4	byte	6	El paquete contiene muestras del canal VIN3 en la primera mitad y muestras del canal VIN4 en la segunda mitad.
DIN_CHANGE	byte	7	Estado de las entradas discretas cambio desde la última lectura.
COUNT1_MATCH	byte	8	Evento "match" del contador 1.
COUNT2_MATCH	byte	9	Evento "match" del contador 2.
VIN5	byte	10	El paquete contiene muestras del canal VIN5.
VIN6	byte	11	El paquete contiene muestras del canal VIN6.

enum UdpStreamPacketIDs : byte: Enumeración (continuación).			
Campos	**Tipo**	**Valor (dec)**	**Descripción**
VIN7	byte	12	El paquete contiene muestras del canal VIN7.
VIN8	byte	13	El paquete contiene muestras del canal VIN8.
VIN_1TO4	byte	14	Contiene muestras de los canales VIN1 a VIN4 en el mismo paquete. Las mismas están distribuidas uniformemente.
VIN_5TO8	byte	15	Contiene muestras de los canales VIN5 a VIN8 en el mismo paquete. Las mismas están distribuidas uniformemente.
VIN5AND6	byte	16	El paquete contiene muestras del canal VIN5 en la primera mitad y muestras del canal VIN6 en la segunda mitad.
VIN7AND8	byte	17	El paquete contiene muestras del canal VIN7 en la primera mitad y muestras del canal VIN8 en la segunda mitad.

9.3 Clase UdpStream

Primero describiremos las clases y objetos de la librería STX8XXX.DLL para manejar paquetes UDP-STREAM y luego, en secciones posteriores, brindaremos ejemplos de uso y aplicación.

La clase *UdpStream* brinda métodos para recibir paquetes UDP-STREAM. El mecanismo consiste en crear un hilo (thread) o tarea (task) que se ejecutara en segundo plano (background) respecto al programa principal (su aplicación) y escuchará el puerto UDP donde el dispositivo STX80XX enviará los paquetes UDP-STREAM.

Cuando lleguen los paquetes, la tarea que se está ejecutando en segundo plano, llamará a un objeto o clase definida por el usuario derivada de la clase *UdpStreamHandle,* que se encargara de procesar el paquete recibido (graficar muestras, guardar muestras en un archivo, tomar una acción de control, etc).

La clase *UdpStreamHandle* es del tipo abstracta, esto quiere decir que esta parcialmente implementada. Solo contiene una estructura principal y no puede ser instanciada. El usuario debe crear su propia clase heredada o derivada de la clase *UdpStreamHandle.*

Cuando usted cree su propia clase derivada de la clase *UdpStreamHandle* deberá implementar dos métodos: RxHandle() y RxStop(). El primer método será llamado cuando el paquete UDP-STREAM sea recibido (allí usted procesara el paquete) y el segundo método será llamado cuando usted decida detener la tarea que escucha paquetes UDP-STREAM, que es útil para terminar operaciones pendientes (cerrar archivos, enviar comandos al dispositivo STX80XX, etc).

Resumiendo, con la clase *UdpStream* crea una tarea en segundo plano para escuchar paquetes UDP-STREAM y con la clase abstracta *UdpStreamHandle* procesa los paquetes recibidos con su propio código.

Ejemplos completos, serán proporcionados en la próximas secciones.

9.3.1 Constructor

UdpStream(UdpStreamHandle OnReceiveHandle): Incializa el objeto.		
Argumentos	**Tipo**	**Descripción**
OnReceiveHandle	E	Objeto del tipo *UdpStreamHandle* que será llamado cuando un paquete UDP-STREAM sea recibido.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
-		

Ejemplo:

```
// Crear objeto del tipo UdpStream, utilizado para operaciones
// de control.
UdpStream UdpStreamControl;

// Crear un objeto del tipo UdpStreamHandle….
// Vea ejemplos en secciones siguientes.

// Instanciar objeto. Le pasamos como parámetro un objeto UdpStreamHandler
// que fue derivado de una clase UdpStreamHandle.
UdpStreamControl = new UdpStream(UdpStreamHandler);
```

9.3.2 Métodos

void StartListening(): Crea una tarea (thread o hilo) en segundo plano para escuchar paquetes UDP-STREAM. Cuando lleguen los paquetes, seran procesados con una clase del usuario derivada de la clase *UdpStreamHandle*.		
Argumentos	**Tipo**	**Descripción**
-		
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		Recuerde llamar al método StopListening() antes de cerrar su aplicación, para eliminar la tarea creada.
2		El dispositivo STX80XX debe ser configurada a través de comandos, para que envíe paquetes UDP-STREAM.

Ejemplo:

```
// Crear tarea en segundo plano para escuchar paquetes.
UdpStreamControl.StartListening();
```

void StopListening(): Elimina la tarea (thread o hilo) en segundo plano que escucha paquetes UDP-STREAM.		
Argumentos	**Tipo**	**Descripción**
-		
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		El método RxStop() de la clase *UdpStreamHandle* será llamado. Ese método puede contener su código para finalizar operaciones pendientes (ejemplo cerrar archivos, enviar algún comando, etc).

Ejemplo:

```
// Eliminar tarea para escuchar paquetes.
UdpStreamControl.StopListening();
```

9.3.3 Enumeraciones

La enumeración UdpStreamPacketIDs fue descripta en secciones anteriores.

A continuación se describe la enumeración UdpStreamStat.

enum UdpStreamStat : int: Código de estado que son devueltos cuando un paquete UDP-STREAM es recibido por la librería.

Campos	Tipo	Valor (dec)	Descripción
Success	int	0	Retorno exitoso.
ErrorTimeout	int	-1	Tiempo de espera agotado esperando el paquete.
ErrorUnknown	int	-2	Otro error. Revisar código de librería.
ErrorIncompleteHeaders	int	-3	El paquete recibido tiene cabeceras incompletas.
ErrorIncompleteData	int	-4	Paquete recibido, pero el contenido de los datos es incompleto.
ErrorSocketException	int	-5	Error del socket.

9.3.4 Clase UdpStreamPacket

class UdpStreamPacket: Esta clase se utiliza para almacenar un paquete UDP-STREAM completo. Los campos relevantes para el usuario se listan a continuación.

Campos	Tipo	Valor (dec)	Descripción
ID	byte	-	Identificador de paquete UDP-STREAM.
DataSize	byte	-	Numero de bytes de datos recibidos.
Data	byte[]	-	Array de bytes que contienen los datos recibidos.

9.4 Clase UdpStreamHandle

Cuando un paquete UDP-STREAM es recibido, esta clase es llamada para procesarlo. Como cada paquete UDP-STREAM puede ser procesado de diferentes formas, ya que es dependiente de la aplicación del usuario, la clase *UdpStreamHandle* debe ser derivada e implementada por el usuario de la librería.

La clase *UdpStreamHandle* es del tipo abstracta, esto quiere decir que esta parcialmente implementada. Solo contiene una estructura principal y no puede ser instanciada. El usuario debe crear su propia clase heredada o derivada de la clase *UdpStreamHandle.*

Cuando usted cree su propia clase derivada de la clase *UdpStreamHandle* deberá implementar dos métodos: **RxHandle()** y **RxStop().** El primer método será llamado cuando el paquete UDP-STREAM sea recibido (allí usted procesará el paquete) y el segundo método será llamado cuando usted decida detener la tarea que escucha paquetes UDP-STREAM, que es útil para terminar operaciones pendientes (cerrar archivos, enviar comandos al dispositivo STX80XX, etc).

La clase *UdpStreamHandle* es muy simple, y tiene la siguiente estructura definida en la librería:

```
public abstract class UdpStreamHandle
{
    public EndPoint DeviceEndPoint = null;

    public int SocketExcepErrorCode = 0;

    public UdpStreamHandle()
    {

    }

    public abstract void RxHandle(UdpStreamPacket Packet, UdpStreamStat PacketStatus);

    public abstract void RxStop();
}
```

Como es posible apreciar en el codigo anterior, la clase *UdpStreamHandle* contiene dos variables publicas y dos métodos.

Los métodos: **RxHandle()** y **RxStop()** son los que usted debe implementar en su clase derivada.

A continuación, explicamos el significado de los métodos y variables de la clase. Luego mostraremos un ejemplo genérico.

9.4.1 Variables

DeviceEndPoint:

Cuando un paquete UDP-STREAM es recibido, aquí se almacena la información de la dirección IP remota y su puerto. La variable es un objeto del tipo EndPoint (más información en la documentación de Microsoft Visual C#).

SocketExcepErrorCode:

Si el paquete recibido devuelve un error UdpStreamStat.ErrorSocketException (ver enumeración UdpStreamStat), en esta variable se deposita el código asociado a ese error. Es del tipo int.

9.4.2 Métodos

public abstract void RxHandle(UdpStreamPacket Packet, UdpStreamStat PacketStatus): Cuando un paquete UDP-STREAM es recibido, este metodo es llamado. Usted debe implementarlo.

Argumentos	**Tipo**	**Descripción**
Packet	E	Paquete UDP-STREAM recibido. Ver clase UdpStreamPacket.
PacketStatus	E	Estado del paquete recibido. Si el paquete fue recibido exitosamente, el valor de esta variable debería ser UdpStreamStat.Success.
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		El método debe implementarse derivando la clase *UdpStreamHandle*.

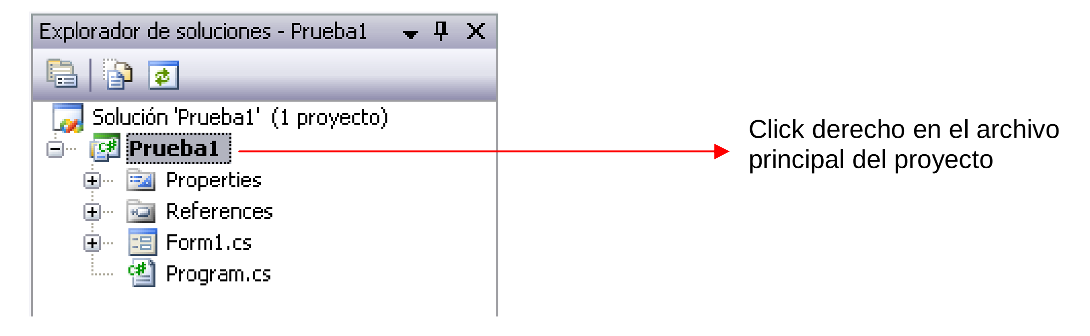
public abstract void RxStop(): Este metodo se llama cuando se dejan de escuchar paquetes UDP-STREAM. Ver metodo StopListening() de clase *UdpStream.*

Argumentos	**Tipo**	**Descripción**
.		
Retorno	**Tipo**	**Descripción**
-		
Notas		**Descripción**
1		El método debe implementarse derivando la clase *UdpStreamHandle*. Aquí puede finalizar operaciones pendientes (cerrar archivos, enviar comandos, etc).

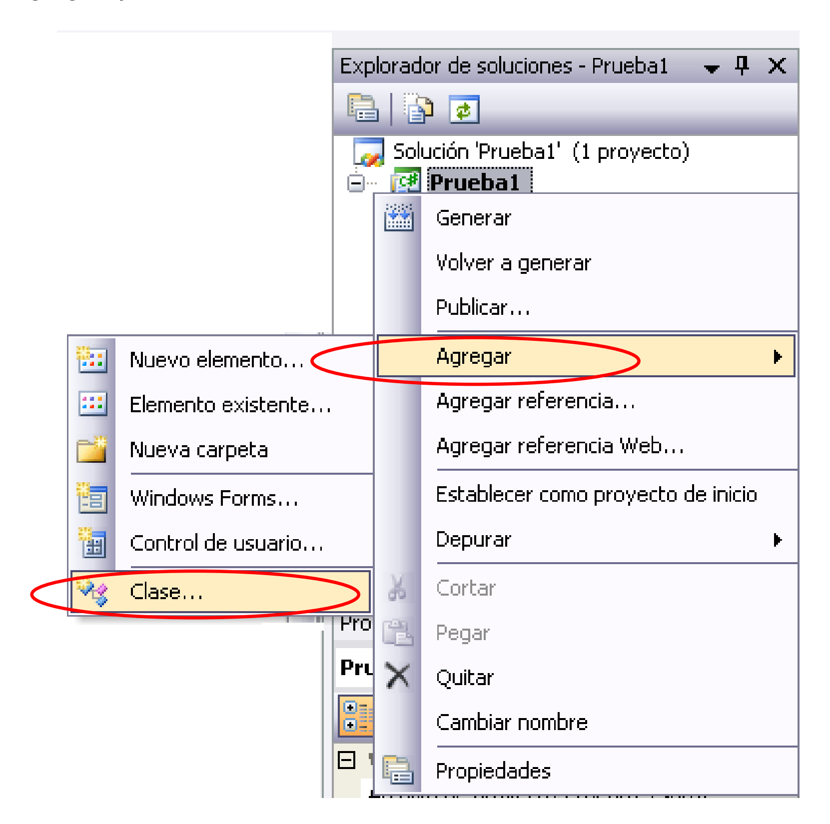
9.4.3 Ejemplo para derivar clase UdpStreamHandle

Para utilizar la clase abstracta *UdpStreamHandle* primero debe derivarla o heredarla en otra clase, lo cual es muy simple. A continuacion mostraremos un ejemplo de una una clase que guarda las muestras recibidas de la entrada analogica Vin1 en un archivo. El ejemplo es a modo orientativo, es decir se utiliza pseudo-codigo para ejemplificar con fines didacticos solamente.

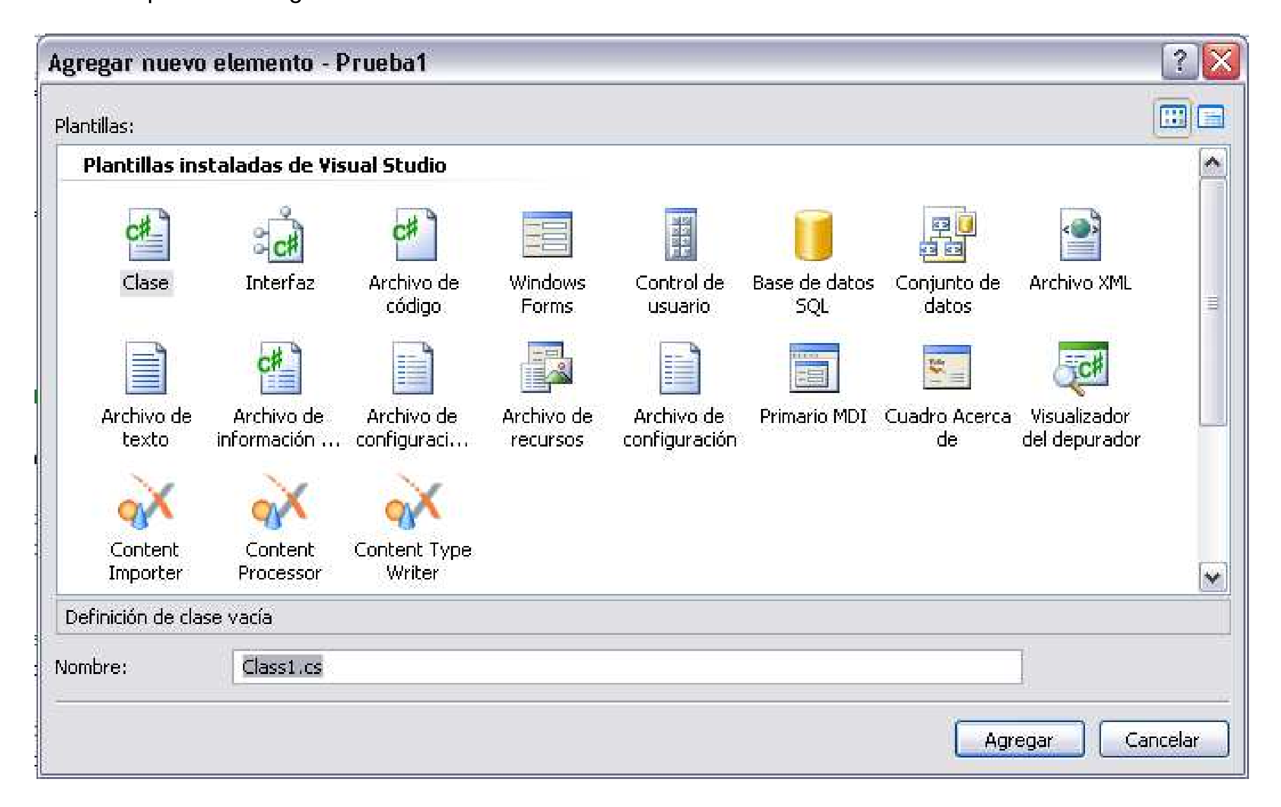
En Microsoft Visual C# para crear una clase, primero debemos tener un proyecto (como se explico al comienzo de este documento) y luego desde el explorador de soluciones, agregamos una clase:



Luego, click en “Agregar” y finalmente click en “Clase”:



Deberia aparecer la siguiete ventana:



Donde dice nombre, escribimos UdpStreamRxFile.cs, que es el nombre que tendrá nuestro objeto y seleccionamos el icono rotulado “Clase”.

Se debería abrir una ventana con el código de nuestra clase creada:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Prueba1
{
   class UdpStreamRxFile
   {
   }
}
```

Notar que la clase está vacía, y no contiene código alguno:

```
   class UdpStreamRxFile
   {
   }
```



Para tener acceso a la librería STX8XXX, debemos agregar la diretiva "using stx8xxx;" es este nuevo archivo:

```
using System;
using System.Collections.Generic;
using System.Text;

using stx8xxx;

namespace Prueba1
{
   class UdpStreamRxFile
   {
   }
}
```

El próximo paso, es derivar la clase a partir de la clase *UdpStreamHandle*, esto se logra anteponiendo dos puntos ":" luego de la declaración de la clase, como se muestra a continuación:

```
   class UdpStreamRxFile : UdpStreamHandle
   {
   }
```

Simple, ¿verdad?

Ahora, solo resta implementar los métodos RxHandle() y RxStop(), utilizando la palabra clave "override".

Recordar que queremos guardar muestras en un archivo, entonces:

```
      public override void RxHandle(UdpStreamPacket Packet, UdpStreamStat PacketStatus)
      {
         // Comprobar si el paquete UDP-STREAM fue recibido con éxito.
         if (PacketStatus == UdpStreamStat.Success)
         {
            if (Packet.ID != (byte) UdpStreamPacketIDs.VIN1)
            {
               // Otro tipo de paquete. Salir.
               return;
            }

            // Guardar datos del paquete en archivo.
            foreach (byte Data in Packet.Data)
            {
               File.Write(Data);
            }
         }
      }
```

Básicamente el método verifica si el paquete fue recibido exitosamente, si pertenece a muestras de canal Vin1 y luego lee los bytes de datos recibidos (Packet.Data) y los guarda en un archivo.
Este método será llamado cada vez que un paquete sea recibido.

Finalmente, el metodo RxStop() lo implementamos como:

```
public override void RxStop()
{
   File.Close();
}
```

El método cerrara el archivo cuando se llame StopListening() desde la clase *UdpStream*.

Finalmente, el pseudo-codigo completo es:

```
using System;
using System.Collections.Generic;
using System.Text;

using stx8xxx;
using System.IO;

namespace Prueba1
{
   class UdpStreamRxFile
   {
      public override void RxHandle(UdpStreamPacket Packet, UdpStreamStat PacketStatus)
      {
         // Comprobar si el paquete UDP-STREAM fue recibido con éxito.
         if (PacketStatus == UdpStreamStat.Success)
         {

            // El paquete recibido contiene muestras del canal Vin1 ?.
            if (Packet.ID != (byte) UdpStreamPacketIDs.VIN1)
            {
               // No, otro tipo de paquete. Salir.
               return;
            }

            // Guardar datos del paquete en archivo.
            foreach (byte Data in Packet.Data)
            {
               File.Write(Data);
            }
         }
      }

      public override void RxStop()
      {
         File.Close();
      }
   }
}
```

Para instanciar la clase creada, desde otro archivo, hacemos:

```
// Crear un objeto "UdpStreamHandler", del tipo "UdpStreamRxFile",
// que fue derivado de una clase abtracta "UdpStreamHandler".
UdpStreamRxFile UdpStreamHandler = new UdpStreamRxFile();
```

En un aplicación real, antes para usar esta clase, debemos escuchar paquetes UDP-STREAM, para ello utilizamos la clase UdpStream y le pasamos la clase derivada:

```
// Crear un objeto "UdpStreamHandler", del tipo "UdpStreamRxFile",
// que fue derivado de una clase abtracta "UdpStreamHandler".
UdpStreamRxFile UdpStreamHandler = new UdpStreamRxFile();

// Crear objeto del tipo UdpStream, utilizado para operaciones
// de control.
UdpStream UdpStreamControl;

// Instanciar objeto. Le pasamos como parametro un objeto UdpStreamHandler
// que fue derivado de una clase UdpStreamHandle.
UdpStreamControl = new UdpStream(UdpStreamHandler);

// Crear tarea en segundo plano para escuchar paquetes.
UdpStreamControl.StartListening();
```

Finalmente, deberiamos enviar un comando al dispositivo STX80XX para activar el sampler y muestrear el canal Vin1:

```
// Se crea un objeto PioBoard del tipo Stx8xxx.
Stx8xxx PioBoard;

// Se crea una instancia de la clase Stx8xxx.
PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);

// Especificar dirección IP de la maquina que recibirá las muestras
// obtenidas por el sampler.
PioBoard.Cmd.BoardConfig.SetUdpStreamHostIp(192, 168, 1, 10);

// Activar sampler A, muestrear canal Vin1 con un Ts = 250 uS,
// equivalente a una frecuencia de muestreo de 4000 Hz.
// Las muestreas se enviaran a través de paquetes UDP-STREAM a la dirección
// IP especificada anteriormente.
PioBoard.Cmd.Vin.SamplerAStart(VinSamplerACh.Vin1, 250);
```

Cada vez que llegue un paquete UDP-STREAM, se llamara al metodo UdpStreamHandler.RxHandle() que definimos en la clase UdpStreamRxFile.

9.4.4 Ejemplo de clase UdpStreamHandle Funcional

La siguiente clase derivada, escribe los paquetes UDP-STREAM en un archivo de texto:

```
class UdpStreamRxFile : UdpStreamHandle
{
   // Archivo para Log.
   private StreamWriter LogFile;

   // Mantiene el numero de paquetes UDP-STREAM recibidos.
   private UInt32 PacketNumber = 0;

   // Constructor: Especifica el nombre del archivo para Log,
   public UdpStreamRxFile(string FileName)
   {
      // Crear archivo Log .
      LogFile = new StreamWriter(FileName, true);
   }

   public override void RxHandle(UdpStreamPacket Packet, UdpStreamStat PacketStatus)
   {
      // Comprobar si el paquete UDP-STREAM fue recibido con éxito.
      if (PacketStatus == UdpStreamStat.Success)
      {
         // Un nuevo paquete recibido.
         PacketNumber++;
         LogFile.WriteLine("Packet Number: {0}", PacketNumber);

         // Guardar información del paquete.
         LogFile.WriteLine("Packet ID / DataSize: {0} / {1}", Packet.ID, Packet.DataSize);

         // Escribir datos del paquete en el archivo.
         foreach (byte Data in Packet.Data)
         {
            LogFile.Write(Data);
            LogFile.Write(" ");
         }

         // Escribir línea vacia...
         LogFile.WriteLine();

         // Forzar a escribir buffer del archivo en disco.
         LogFile.Flush();
      }
   }

   public override void RxStop()
   {
      LogFile.Flush();
      LogFile.Close();
   }
}
```

Para utilizarla, hay que agregar en el archivo de la clase, las siguientes directivas “using”:

```
using System.IO;
using stx8xxx;
```

La primera directiva es para las funciones de archivo, y la segunda es para la librería STX8XXX.DLL.

Para instanciarla desde otra clase, hacemos:

```
// Crear un objeto "UdpStreamHandler" que guardara los paquetes UDP-STREAM
// recibidos en un archivo de texto llamado "log.txt".
UdpStreamRxFile UdpStreamHandler = new UdpStreamRxFile("log.txt");
```

Cuando lleguen los paquetes, el archivo “log.txt” será escrito con el método RxHandle() y tendrá la siguiente estructura:

Packet Number: 1
Packet ID / Size: 1 / 127
0 93 3 1 0 0 3 0 3 …………………………………………..

Packet Number: 2
Packet ID / Size: 1 / 127
0 1 3 1 5 0 3 0 3 …………………………………………..

Las primeras dos líneas muestra el numero de paquetes recibidos, el identificador de paquete (ID) y la cantidad de datos recibidos (DataSize). La tercera línea muestra el valor de los bytes de datos recibidos. En este caso se recibieron dos paquetes con muestras del canal Vin1.

10 Manipulando Paquetes UDP-STREAM

En esta sección se explicara como configurar el dispositivo STX80XX para enviar paquetes UDP-STREAM y ejemplos para manipularlos desde su programa.

En la sección anterior, se pudo observar que la metodología para trabajar con paquetes UDP-STREAM es la siguiente:

1. Crear un Objeto Stx8xxx para el dispositivo STX80XX con comandos.
2. Crear un Objeto UdpStream para poder recibir paquetes UDP-STREAM.
3. Crear un Objeto derivado de la clase UdpStreamHandle, para poder procesar cada paquete UDP-STREAM recibido.
4. Enviar al dispositivo STX80XX un comando para especificar la dirección IP de a máquina a donde deben enviarse los paquetes UDP-STREAM.
5. Enviar al dispositivo STX80XX un comando para que el mismo envíe paquetes UDP-STREAM, por ejemplo: muestrear entradas analógicas, valor “match” de los contadores, etc…

A continuacion, explicaremos como activar los paquetes UDP-STREAM disponibles en el dispositivo STX80XX. Se recomienda leer la seccion anterior con detenimiento, antes de continuar.

Los paquetes UDP-STREAM estan disponibles para las siguientes caracteristicas:

- Entradas Discretas: Se avisa cuando alguna entrada digital (DIN1, DIN2, etc) cambia de estado.
- Contadores: Se avisa cuando un contador llego a un valor determinado (valor match).
- Entradas Analógicas: Se envían las muestras obtenidas de entradas analógicas.

10.1 Entradas Discretas, Paquetes UDP-STREAM

En esta sección explicaremos como manipular paquetes que informan que alguna entrada digitales DIN1 a DIN8 cambio su estado desde la última lectura. Este ejemplo puede adaptarse para leer menor o mayor cantidad de entradas digitales.

Para ello, crearemos un nuevo proyecto en Microsoft Visual C#, llamado "**Din_UdpStream**". **El código completo del proyecto se adjunta como ejemplo en este documento, en un archivo separado y comprimido.**

Un paquete UDP-STREAM que informa sobre el estado de las entradas discretas, tiene la siguiente estructura:

ID=7	DataSize=1	Data Bytes = DinState

Donde el identificador de paquete (ID), vale 7 y la cantidad de bytes de datos (DataSize) es 1. El único byte de dato (DataBytes) contiene una sola variable de 8-bit, que llamaremos DinState (estado de entrada discretas).

La variable DinState transmitida, tiene 8-bit, es decir es un byte, y cada bit representa el estado de las 8 entradas discretas:

Byte de Dato Recibido (DinState)							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIN8	DIN7	DIN6	DIN5	DIN4	DIN3	DIN2	DIN1

Cuando algún bit este en "1", la entrada discreta estará polarizada o en nivel alto. De lo contrario se mantiene en "0".

El programa de prueba "**Din_UdpStream**" consistirá en una ventana que muestre el estado de las 8 entradas discretas y cuando llegue algún paquete UDP-STREAM con información sobre las entradas, deberá actualizar sus estados en pantalla.

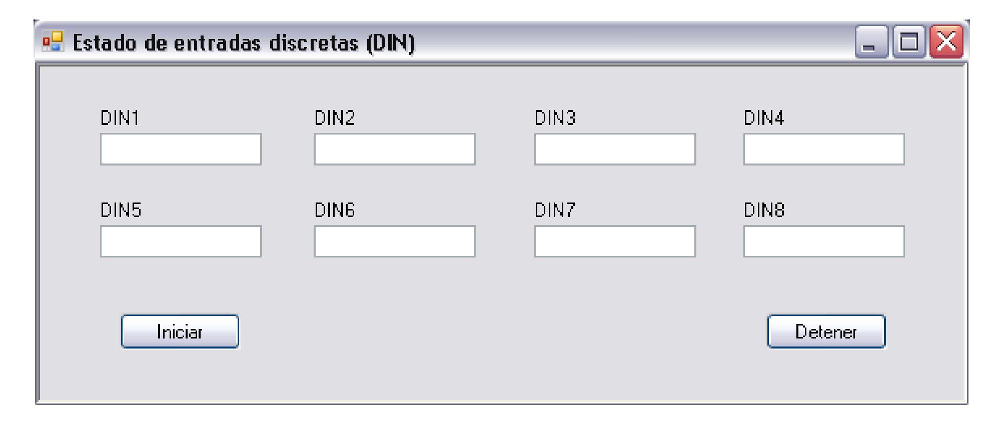
El procedimiento es el siguiente:

1. Crear ventana grafica.
2. Crear código para manejar paquetes UDP-STREAM.
3. Enviar al dispositivo los comandos necesarios para que informe el estado de las entradas discretas a nuestra dirección IP.

Nota: Consulte la hoja de datos del dispositivo para mayor exactitud, pero en general un dispositivo STX80XX muestrea el estado de las entradas digitales en un intervalo mínimo de 50 mS. Esto quiere decir que, señales con ancho de pulso inferior a 50mS no generan paquetes UDP-STREAM.

10.2 Crear Ventana

La ventana consiste en 8 TextBoxs, con el estado de las entradas discretas y dos botones. Un botón para iniciar el monitoreo de los paquetes que llegan y otro botón para detener la operación.



Cuando se reciba el estado de las entradas, las TextBox mostraran “Alto” o “Bajo”, según el estado de las entradas.

10.2.1 Crear Código para Manejar Paquetes UDP-STREAM

Primero, **debemos incluir la librería STX8XXX.DLL** en nuestro proyecto, como se mostró en el inicio del documento.

Luego creamos una clase llamada UdpStreamRxDin, que debemos derivar de la clase UdpStreamHandle. A la clase la guardamos en un archivo llamado UdpStreamRxDin.cs.

La clase UdpStreamRxDin, a través del método RxHandle(), actualizara el valor de los “TextBox” cada vez que llegue un paquete UDP-STREAM con el estado de las entradas.

Contiene dos variables globales, mainForm (que permite acceder a los objetos del formulario principal) y pioBoard (permite acceder al objeto para controlar el dispositivo STX80XX). Las variables, serán inicializadas cuando se instancie la clase UdpStreamRxDin desde el formulario principal, a través de su constructor.

La clase UdpStreamRxDin es la siguiente:

```
class UdpStreamRxDin : UdpStreamHandle
{
    /// <summary>
    /// Referencia al formulario principal.
    /// </summary>
    private Form1 mainForm = null;

    /// <summary>
    /// Referencia al objeto para controlar el dispositivo STX80XX.
    /// </summary>
    private Stx8xxx pioBoard = null;
```

```
/// <summary>
/// Inicializa la clase.
/// </summary>
/// <param name="MainForm">Referencia al formulario principal Form1.</param>
/// <param name="PioBoard">Referencia al objeto para controlar el dispositivo STX80XX.</param>
public UdpStreamRxDin(Form1 MainForm, Stx8081 PioBoard)
{
   // Guardamos la referencia al formulario principal.
   mainForm = MainForm;

   // Guardamos la referencia para controlar el dispositivo STX80XX.
   pioBoard = PioBoard;
}

public override void RxHandle(UdpStreamPacket Packet, UdpStreamStat PacketStatus)
{
   // Obtenemos el identificador de paquete recibido.
   UdpStreamPacketIDs ID = (UdpStreamPacketIDs) Packet.ID;

   // Comprobar si el paquete UDP-STREAM fue recibido con exito.
   if (PacketStatus == UdpStreamStat.Success)
   {
      // Comprobar, si el paquete tiene informacion del
      // estado de la entradas discretas.
      if (ID == UdpStreamPacketIDs.DIN_CHANGE)
      {
         // Guardamos el estado de las entradas discretas recibido,
         // el cual se encuentra en el byte 0, de los datos recibidos.
         byte DinStat = Packet.Data[0];

         // ACTUALIZAMOS VALOR DE LAS TEXTBOXs CON EL ESTADO DE
         // LAS ENTRADAS DISCRETAS.
         // NOTAR QUE NO ENVIAMOS COMANDOS AL DISPOSITIVO. SOLO LEEMOS
         // LA VARIABLE DinStat.

         if(pioBoard.Cmd.Din.CheckIfHigh(DinStat, DinInput.Din1))
         {
            mainForm.textBox1.Text = "Alto";
         }
         else
         {
            mainForm.textBox1.Text = "Bajo";
         }

         if (pioBoard.Cmd.Din.CheckIfHigh(DinStat, DinInput.Din2))
         {
            mainForm.textBox2.Text = "Alto";
         }
```

```
else
{
   mainForm.textBox2.Text = "Bajo";
}

if (pioBoard.Cmd.Din.CheckIfHigh(DinStat, DinInput.Din3))
{
   mainForm.textBox3.Text = "Alto";
}
else
{
   mainForm.textBox3.Text = "Bajo";
}

if (pioBoard.Cmd.Din.CheckIfHigh(DinStat, DinInput.Din4))
{
   mainForm.textBox4.Text = "Alto";
}
else
{
   mainForm.textBox4.Text = "Bajo";
}

if (pioBoard.Cmd.Din.CheckIfHigh(DinStat, DinInput.Din5))
{
   mainForm.textBox5.Text = "Alto";
}
else
{
   mainForm.textBox5.Text = "Bajo";
}

if (pioBoard.Cmd.Din.CheckIfHigh(DinStat, DinInput.Din6))
{
   mainForm.textBox6.Text = "Alto";
}
else
{
   mainForm.textBox6.Text = "Bajo";
}

if (pioBoard.Cmd.Din.CheckIfHigh(DinStat, DinInput.Din7))
{
   mainForm.textBox7.Text = "Alto";
}
else
{
   mainForm.textBox7.Text = "Bajo";
}
```

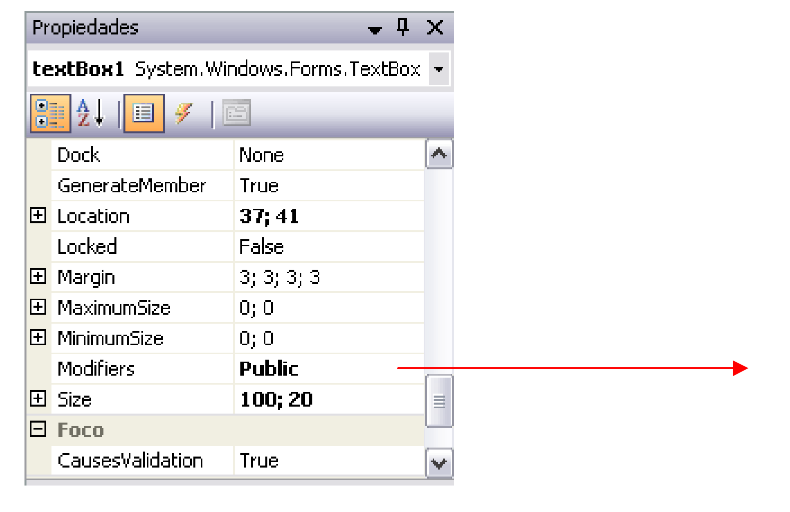
```
                if (pioBoard.Cmd.Din.CheckIfHigh(DinStat, DinInput.Din8))
                {
                    mainForm.textBox8.Text = "Alto";
                }
                else
                {
                    mainForm.textBox8.Text = "Bajo";
                }
            }
        }
    }

    public override void RxStop()
    {
        // No enviar paquetes UDP-STREAM informando
        // cambios de estados de entradas discretas.
        pioBoard.Cmd.Din.InformChanges(false);
    }
}
```

Cuando instancie esta clase desde el formulario principal (Form1.cs), deberá pasar los objetos necesarios para acceder al formulario principal y al dispositivo STX80XX desde esta clase, de la siguiente forma:

```
// Instanciar Manejador de paquetes UDP-STREAM.
// Pasamos como referencia el formulario principal (this)
// y el objeto para controlar el dispositivo STX80XX en su argumentos.
UdpStreamHandler = new UdpStreamRxDin(this, PioBoard);
```

Y para que los objetos **TextBox** sean visibles, deben ser públicos, entonces, en el entorno de desarrollo Microsoft Visual C#, vamos a la ventana de diseño del formulario principal y seleccionamos un **TextBox**. Luego vamos a la ventana de propiedades, y en el campo "**Modifiers**", seleccionamos "**Public**". Repetimos la operación con todos los "**TextBox**" del formulario.



Seleccionamos "**Modifiers**" y elegimos "**Public**" para hacer visible externamente (publico) la **TextBox**.

Ver código de ejemplo adjuntado al documento.



10.2.2 Enviar Comandos al Dispositivo

A continuación, vamos a definir el constructor del formulario Form1.cs y los eventos click para el botón "**Iniciar**" y el botón "**Detener**" del formulario.

VARIABLES GLOBALES:

Objetos accesibles desde toda la clase Form1:

```
/// <summary>
/// Objeto para controlar el dispositivo STX80XX.
/// Power I/O Board.
/// </summary>
private Stx8xxx PioBoard;

/// <summary>
/// Objeto para controlar paquetes UDP-STREAM.
/// </summary>
private UdpStream UdpStreamControl = null;

/// <summary>
/// Manejador de paquetes UDP-STREAM.
/// </summary>
private UdpStreamRxDin UdpStreamHandler = null;
```

CONSTRUCTOR:

Aquí inicializamos variables y objetos del formulario principal:

```
// Constructor del programa.
public Form1()
{
   // Requerido por Visual C#.
   InitializeComponent();

   // Instanciar objeto para controlar el dispositivo STX80XX.
   // Pasamos direccion IP, password y modelo de dispositivo utilizado.
   PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);

   // Instanciar Manejador de paquetes UDP-STREAM.
   // Pasamos como referencia el formulario principal (this)
   // y el objeto para controlar el dispositivo STX80XX en su argumentos.
   UdpStreamHandler = new UdpStreamRxDin(this, PioBoard);

   // Instanciar objeto para controlar paquetes UDP-STREAM.
   // Le pasamos como argumento el manejador de paquetes que
   // creamos para obtener estado de entradas discretas.
   UdpStreamControl = new UdpStream(UdpStreamHandler);
}
```

BOTON INICIAR (EVENTO CLICK):

Al hacer click sobre este botón, el programa debería escuchar paquetes UDP-STREAM y enviarle al dispositivo STX80XX la dirección IP de nuestra maquina donde enviará los paquetes con la información del estado de las entradas digitales. Por defecto, se utilizó la dirección “192.168.1.11” como dirección de nuestra maquina, puede cambiarla a la dirección correcta de su máquina y luego recompile el código.

```
// Iniciar Escucha de paquetes UDP-STREAM.
private void button1_Click(object sender, EventArgs e)
{
   // Primero le especificamos al dispositivo STX80XX la direccion
   // IP de nuestra computadora, a la cual debe mandar los paquetes UDP-STREAM.
   PioBoard.Cmd.BoardConfig.SetUdpStreamHostIp(192, 168, 1, 11);

   // Configuramos la libreria para escuchar paquetes UDP-STREAM
   // en segundo plano.
   UdpStreamControl.StartListening();

   // Le especificamos al dispositivo STX80XX, que informe
   // el cambio de las entradas discretas a traves de paquetes
   // UDP-STREAM.
   PioBoard.Cmd.Din.InformChanges(true);
}
```

BOTON DETENER (EVENTO CLICK):

Cuando hacemos click en este botón, nuestra computadora deja de escuchar paquetes UDP-STREAM:

```
// Detener Escucha de paquetes UDP-STREAM.
private void button2_Click(object sender, EventArgs e)
{
   // Al ejecutar este metodo, el metodo UdpStreamHandler.RxStop()
   // deberia ser llamado tambien, y el dispositivo dejaria de enviar
   // el estado de las entradas discretas.
   UdpStreamControl.StopListening();
}
```

10.2.3 Funcionamiento

Utilice el código de ejemplo provisto, compile el programa y pruébelo. No se comprobaran errores (por fines didácticos) por lo tanto verifique que el dispositivo esté conectado y funcionando correctamente, así como las direcciones IP empleadas sean las correctas.

Inicie la escucha de paquetes UDP-STREAM, haciendo click en el botón “Iniciar”.

Cuando el dispositivo STX80XX detecte un cambio en sus entradas discretas, el programa deberá actualizar el estado de las entradas, y la ventana principal podrá verse como se muestra a continuación:



En la ventana se observa el programa funcionando. Las entradas DIN1 a DIN6 se encuentran en nivel “Bajo”. Las entradas DIN7 y DIN8 pasaron del nivel “Bajo” al nivel “Alto”. Este cambio fue detectado por el dispositivo y envió a la computadora un paquete UDP-STREAM con el nuevo estado de las entradas. El programa recibió el paquete y posteriormente mostró el estado de las entradas.

Antes de cerrar el programa, recuerde hacer click en el botón “Detener”.

10.3 Contadores, Paquetes UDP-STREAM

En esta sección explicaremos como manipular paquetes que informan si algún contador (COUNT1 o COUNT2) ha llegado al valor “match” establecido por el usuario. Si el contador de eventos alcanza el valor “match”, el dispositivo STX80XX informa a través de un paquete UDP-STREAM la ocurrencia.

Cuando el valor “match” es alcanzado, el contador es puesto a cero nuevamente.

Crearemos un nuevo proyecto en Microsoft Visual C#, llamado “**Count_UdpStream**”. **El código completo del proyecto se adjunta como ejemplo en este documento, en un archivo separado y comprimido.**

Un paquete UDP-STREAM que informa sobre el evento “match” de algún contador, tiene la siguiente estructura:

ID	DataSize	Data Bytes
X	4	MatchValue0 | MatchValue1 | MatchValue2 | MatchValue3

Donde el identificador de paquete (ID), puede valer 8 (match del COUNT1) o 9 (match del COUNT2).

La cantidad de bytes de datos (DataSize) es 4. Esto es debido a que se transmite el valor “match” del contador, que tiene 32 bits o 4 bytes.

Los datos (DataBytes) contienen 4 bytes, que representan el valor “match” del contador que produjo el evento. Los 4 bytes representan una variable de 32-bits, el primer byte recibido tiene los 8-bits menos significativos.

Se puede utilizar el método “estático”, BytesToInt() que provee la librería STX8XXX dentro de la clase “Tools”, para convertir los 4 bytes recibidos en una variable de 32-bits, llamada “MatchValue”:

```
UInt32 MatchValue;

MatchValue = Tools.BytesToInt(Packet.Data, 0);
```

Otra alternativa, seria utilizar desplazamientos “<<”:

```
MatchValue  = ((UInt32) Packet.Data[0]);
MatchValue |= ((UInt32) Packet.Data[1]) << 8;
MatchValue |= ((UInt32) Packet.Data[2]) << 16;
MatchValue |= ((UInt32) Packet.Data[3]) << 24;
```

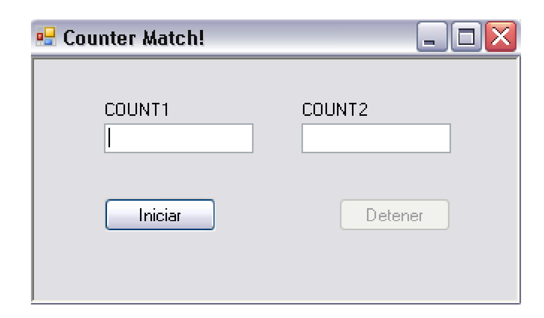
El programa de prueba “**Count_UdpStream**” consistirá en una ventana que muestre si algún contador llego al valor “match”. Cuando un paquete UDP-STREAM llegue informando el evento “match”, una “TextBox” informara del valor “match” recibido.

El procedimiento es el siguiente:

1. Crear ventana grafica.
2. Crear código para manejar paquetes UDP-STREAM.
3. Enviar al dispositivo los comandos necesarios para que informe el evento “match” a nuestra dirección IP.

10.3.1 Crear Ventana

La ventana consiste en dos **TextBox** y dos botones. Las **TextBox** serán completadas con el valor "match" de los contadores (COUNT1 o COUNT2), cuando alcancen la cuenta definida por el usuario y sea informado a través de un paquete UDP-STREAM al programa.



Al hacer clicks en el botón "Iniciar", el programa configura los contadores y quedara a la espera de paquetes UDP-STREAM. Al hacer click en el botón "Detener", el programa detiene la escucha de paquete y desactiva los contadores en el dispositivo.

10.3.2 Crear código para manejar paquetes UDP-STREAM

Primero, **debemos incluir la librería STX8XXX.DLL** en nuestro proyecto, como se mostró en el inicio del documento.

Luego creamos una clase llamada UdpStreamRxCount, que debemos derivar de la clase UdpStreamHandle. A la clase la guardamos en un archivo llamado UdpStreamRxCount.cs.

La clase UdpStreamRxCount, a través del método RxHandle(), actualizara el valor de los "TextBox" cada vez que llegue un paquete UDP-STREAM con el evento "match" de los contadores.

Contiene dos variables globales, mainForm (que permite acceder a los objetos del formulario principal) y pioBoard (permite acceder al objeto para controlar el dispositivo STX80XX). Las variables serán inicializadas cuando se instancie la clase UdpStreamRxCount desde el formulario principal, a través de su constructor.

La clase UdpStreamRxCount es la siguiente:

```
class UdpStreamRxCount : UdpStreamHandle
{
    /// <summary>
    /// Referencia al formulario principal.
    /// </summary>
    private Form1 mainForm = null;

    /// <summary>
    /// Referencia al objeto para controlar el dispositivo STX80XX.
    /// </summary>
    private Stx8xxx pioBoard = null;
```

```
    /// <summary>
    /// Inicializa la clase.
    /// </summary>
    /// <param name="MainForm">Referencia al formulario principal Form1.</param>
    /// <param name="PioBoard">Referencia al objeto para controlar el dispositivo STX80XX.</param>
    public UdpStreamRxCount(Form1 MainForm, Stx8xxx PioBoard)
    {
        // Guardamos la referencia al formulario principal.
        mainForm = MainForm;

        // Guardamos la referencia para controlar el dispositivo STX80XX.
        pioBoard = PioBoard;
    }

    public override void RxHandle(UdpStreamPacket Packet, UdpStreamStat PacketStatus)
    {
        // Obtenemos el identificador de paquete recibido.
        UdpStreamPacketIDs ID = (UdpStreamPacketIDs) Packet.ID;

        // Comprobar si el paquete UDP-STREAM fue recibido con exito.
        if (PacketStatus == UdpStreamStat.Success)
        {
            // Comprobar, si el paquete tiene informacion del contador 1.
            if (ID == UdpStreamPacketIDs.COUNT1_MATCH)
            {
                // Obtenemos el valor "match"
                UInt32 MatchValue1 = Tools.BytesToInt(Packet.Data, 0);

                // Mostramos el valor "match" en la TextBox.
                mainForm.textBox1.Text = MatchValue1.ToString();
            }

            // Comprobar, si el paquete tiene informacion del contador 2.
            if (ID == UdpStreamPacketIDs.COUNT2_MATCH)
            {
                // Obtenemos el valor "match"
                UInt32 MatchValue2 = Tools.BytesToInt(Packet.Data, 0);

                // Mostramos el valor "match" en la TextBox.
                mainForm.textBox2.Text = MatchValue2.ToString();
            }
        }
    }

    public override void RxStop()
    {
        // Desactivar contadores.
        // El dispositivo no informará mas los eventos "match".
        pioBoard.Cmd.Count.Count1Disable();
        pioBoard.Cmd.Count.Count2Disable();
    }
}
```

Cuando instancie esta clase desde el formulario principal (Form1.cs), deberá pasar los objetos necesarios para acceder al formulario principal y al dispositivo STX80XX desde esta clase, de la siguiente forma:

```
// Instanciar Manejador de paquetes UDP-STREAM.
// Pasamos como referencia el formulario principal (this)
// y el objeto para controlar el dispositivo STX80XX en su argumentos.
UdpStreamHandler = new UdpStreamRxCount(this, PioBoard);
```

Y para que los objetos **TextBox** sean visibles, deben ser públicos. Para ello desde el entorno de desarrollo Microsoft Visual C#, vamos a la ventana de diseño del formulario principal y seleccionamos un **TextBox**. Luego vamos a la ventana de propiedades, y en el campo "**Modifiers**", seleccionamos "**Public**". Repetimos la operación con todos los "**TextBox**" del formulario. Este paso se explico con más detalle, en la sección de manejo para paquetes de entradas discretas.

10.3.3 Enviar Comandos al Dispositivo

A continuación, vamos a definir el constructor del formulario Form1.cs y los eventos click para el botón "Iniciar" y el botón "Detener" del formulario.

VARIABLES GLOBALES:

Objetos accesibles desde toda la clase Form1:

```
/// <summary>
/// Objeto para controlar el dispositivo STX80XX.
/// Power I/O Board.
/// </summary>
private Stx8xxx PioBoard;

/// <summary>
/// Objeto para controlar paquetes UDP-STREAM.
/// </summary>
private UdpStream UdpStreamControl = null;

/// <summary>
/// Manejador de paquetes UDP-STREAM.
/// </summary>
private UdpStreamRxCount UdpStreamHandler = null;
```

CONSTRUCTOR:

Aquí inicializamos variables y objetos del formulario principal:

```
// Constructor del programa.
public Form1()
{
    // Requerido por Visual C#.
    InitializeComponent();
```



```
        // Instanciar objeto para controlar el dispositivo STX80XX.
        // Pasamos direccion IP, password y modelo de dispositivo utilizado.
        PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);

        // Instanciar Manejador de paquetes UDP-STREAM.
        // Pasamos como referencia el formulario principal (this)
        // y el objeto para controlar el dispositivo STX80XX en su argumentos.
        UdpStreamHandler = new UdpStreamRxCount(this, PioBoard);

        // Instanciar objeto para controlar paquetes UDP-STREAM.
        // Le pasamos como argumento el manejador de paquetes que
        // creamos para recibir los eventos "match" de los contadores.
        UdpStreamControl = new UdpStream(UdpStreamHandler);

        // Deshabilitar boton2 y habilitar boton1.
        button1.Enabled = true;
        button2.Enabled = false;
    }
```

BOTON INICIAR (EVENTO CLICK):

Al hacer click sobre este botón, el programa debería escuchar paquetes UDP-STREAM y enviarle al dispositivo STX80XX la dirección IP de nuestra maquina donde el mismo debe enviar los paquetes con la información del evento “match” de los contadores. Por defecto, se utilizo la dirección “192.168.1.11” como dirección de nuestra maquina, cámbiela a la dirección correcta de su máquina, y luego recompile el código.

```
    // Iniciar Escucha de paquetes UDP-STREAM.
    private void button1_Click(object sender, EventArgs e)
    {
        // Primero le especificamos al dispositivo STX80XX la dirección
        // IP de nuestra computadora, a la cual debe mandar los
        // paquetes UDP-STREAM.
        PioBoard.Cmd.BoardConfig.SetUdpStreamHostIp(192, 168, 1, 11);

        // Configuramos la librería para escuchar paquetes UDP-STREAM
        // en segundo plano.
        UdpStreamControl.StartListening();

        // Activamos Contadores COUNT1 y COUNT2.
        // Luego establecemos valores "match".

        // COUNT1: Activar cuenta de eventos por flanco ascendente (Rising Edge).
        PioBoard.Cmd.Count.Count1Enable(CountEdge.Rising);

        // COUNT1: Cuando el contador de eventos alcance el número 100,
        // avisara a una dirección IP a través de un paquete
        // UDP-STREAM la ocurrencia de un evento "match".
        PioBoard.Cmd.Count.Count1SetMatch(100);
```

```
        // COUNT2: Activar cuenta de eventos por flanco ascendente (Rising Edge).
        PioBoard.Cmd.Count.Count2Enable(CountEdge.Rising);

        // COUNT2: Cuando el contador de eventos alcance el número 3000,
        // avisara a una dirección IP a través de un paquete
        // UDP-STREAM la ocurrencia de un evento "match".
        PioBoard.Cmd.Count.Count2SetMatch(3000);

        // Deshabilitar boton1 y habilitar boton2.
        button1.Enabled = false;
        button2.Enabled = true;
    }
```

BOTON DETENER (EVENTO CLICK):

Cuando hacemos click en este botón, nuestra computadora deja de escuchar paquetes UDP-STREAM:

```
    // Detener Escucha de paquetes UDP-STREAM.
    private void button2_Click(object sender, EventArgs e)
    {
        // Al ejecutar este método, el método UdpStreamHandler.RxStop()
        // debería ser llamado también, y el dispositivo dejaría de enviar
        // los eventos "match" de los contadores.
        UdpStreamControl.StopListening();

        // Deshabilitar boton2 y habilitar boton1.
        button1.Enabled = true;
        button2.Enabled = false;
    }
```

10.3.4 Funcionamiento

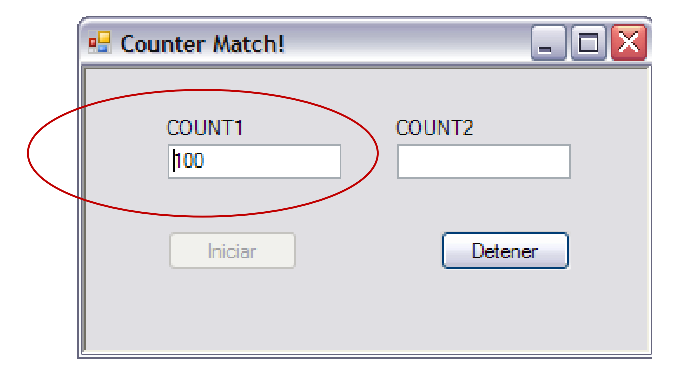
Utilice el código de ejemplo provisto, compile el programa y pruébelo. No se comprobaran errores (por fines didácticos) por lo tanto verifique que el dispositivo esté conectado y funcionando correctamente, así como las direcciones IP empleadas sean las correctas.

Se recomienda conectar un generador de onda cuadrada (100 Hz, 12V pico, duty cycle 50%) a las entradas COUNT1 y COUNT2 del dispositivo (consulte hoja de datos para localizarlas) así los contadores empezarán a contar.

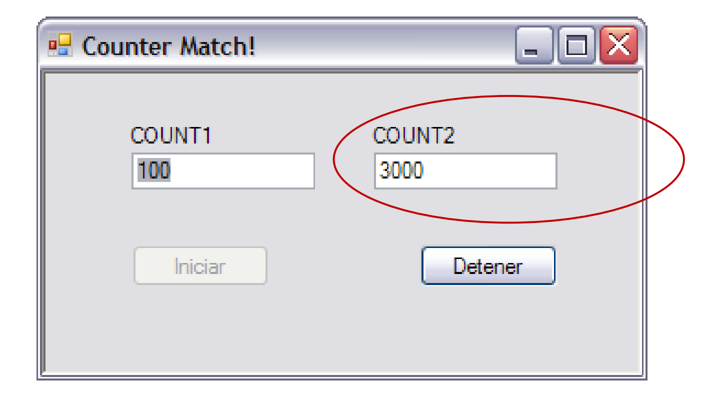
Inicie la escucha de paquetes UDP-STREAM, haciendo click en el botón “Iniciar”.



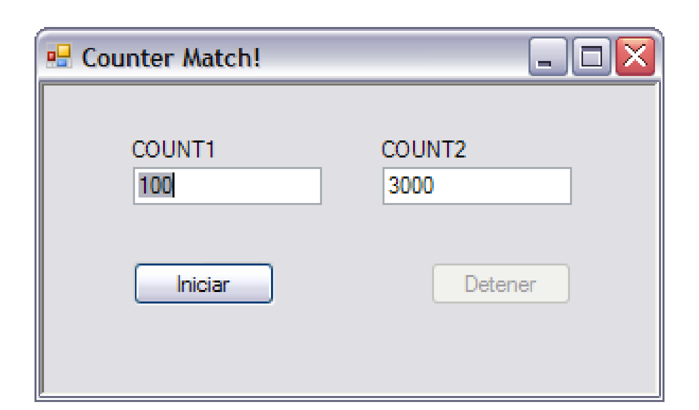
Cuando el contador 1 llegue a la cuenta de 100, el dispositivo enviará un paquete con el evento “match” y el programa mostrara la cuenta recibida, de la siguiente forma:



Igualmente, cuando el contador 2, llegue a la cuenta de 3000, es decir 30 segundos después si la frecuencia de entrada es de 100 Hz, el dispositivo enviará un paquete con el evento “match” y el programa mostrará la cuenta recibida de la siguiente forma:



Antes de cerrar el programa, recuerde hacer click en el botón “Detener”, para cancelar el envío de paquetes UDP-STREAM desde el dispositivo y detener la escucha de paquetes en el programa.



10.4 Entradas Analógicas, Paquetes UDP-STREAM

En esta sección explicaremos como manipular paquetes que contienen muestras de alguna entrada analógica del dispositivo STX80XX.

Según el modelo adquirido, el dispositivo contiene uno o dos samplers (muestreadotes) en general. Consulte hoja de datos para detalles. Por ejemplo, en el modelo STX8081, el sampler A, puede muestrear las entradas VIN1 y/o VIN2 y el sampler B, puede muestrear las entradas VIN3 y/o VIN4.

Cuando activamos un sampler, el dispositivo obtiene las muestras de un canal a una tasa de muestreo fija. Las muestras son almacenadas en un buffer interno del dispositivo, capaz de almacenar 127 muestras de 8-bits cada una. Al llenarse el buffer de muestras, el dispositivo envía las muestras en un paquete UDP-STREAM a la computadora.

Un solo paquete UDP-STREAM puede contener muestras de una entrada analógica (ej: VIN1) o de dos entradas analógicas (ej: VIN1 y VIN2) al mismo tiempo.

Cuando las muestras de dos entradas analógicas son enviadas al mismo tiempo, la primera mitad de datos (127 bytes) pertenecen a un canal y la segunda mitad (restantes 127 bytes) pertenecen al otro canal.

Un paquete UDP-STREAM que contiene muestras de una entrada analógica, tiene la siguiente estructura:

ID	DataSize	Data Bytes
X	127 o 254	Muestra[0], Muestra[1], … , Muestra[DataSize-1]

El identificador de paquete (ID) nos indicara el tipo de muestra contenida en el paquete. En el caso de contener muestras de un solo canal, su valor es VIN1, VIN2, VIN3 o VIN4. El campo DataSize, en este caso, tiene el valor numérico de 127. El campo de datos “DataBytes” contiene muestras de la entrada (Muestra[0] a Muestra[126]).

Si ID contiene el valor VIN1AND2 o VIN3AND4, el paquete contiene muestras de dos entradas analógicas al mismo tiempo. El campo DataSize, en este caso, tiene el valor numérico de 254. La primera mitad de los datos contiene muestras de la entrada más baja (Muestra[0] a Muestra[126]) y la segunda mitad de los datos contiene muestras de la entrada más alta (Muestra[127] a Muestra[253]).

Para el programa de prueba, crearemos un nuevo proyecto en Microsoft Visual C#, llamado “**Vin_UdpStream**”, el cual almacenará en un archivo de texto, las muestras obtenidas de la entrada analógica VIN1. Los datos en el archivo de salida estarán en formato CSV, así fácilmente puede ser importado al Excel para graficar las muestras.

El código completo del proyecto se adjunta como ejemplo en este documento, en un archivo separado y comprimido.

El procedimiento para crear el programa, es el siguiente:

1. Crear ventana grafica.
2. Crear código para manejar paquetes UDP-STREAM.
3. Enviar al dispositivo los comandos necesarios para que envíe las muestras a nuestra dirección IP.

10.4.1 Crear Ventana

El programa consiste en una ventana, donde es posible especificar el nombre del archivo en el cual se almacenarán las muestras una vez adquiridas. También, existe una caja numérica (**NumericUpDown**) que permite especificar la cantidad máxima de muestras que deben ser adquiridas.



Cuando se hace click en el botón "**Iniciar**", el programa crea el archivo para almacenar las muestras y activa el sampler A del dispositivo. Las muestras serán adquiridas y enviadas mediante paquetes UDP-STREAM a la computadora.

Al recibir los paquetes (cada uno con 127 muestras del canal VIN1), el programa almacena las muestras en el archivo especificado (c/u en una línea separada), por ejemplo "vin1.txt".

Luego, el archivo puede ser leído o graficado desde Microsoft Excel si se importa como "CSV".

Finalmente si la cantidad de muestras almacenadas excede el permitido, el programa, termina la operación y desactiva el sampler A del dispositivo.

El usuario debería hacer click en "Detener" si no desea adquirir más muestras o luego de que el programa termine de adquirir los datos.

Una barra de progreso indicara gráficamente el porcentaje de muestras adquiridas.

10.4.2 Crear código para manejar paquetes UDP-STREAM

Primero, **debemos incluir la librería STX8XXX.DLL** en nuestro proyecto, como se mostró en el inicio del documento.

Luego creamos una clase llamada UdpStreamRxVin, que debemos derivar de la clase UdpStreamHandle. A la clase la guardamos en un archivo llamado UdpStreamRxVin.cs.

La clase UdpStreamRxVin, a través del método RxHandle(), actualizara la barra de progreso (ProgressBar) y almacenara las muestras en un archivo, cada vez que llegue un paquete UDP-STREAM con las muestras de la entrada VIN1.

Contiene varias variables globales, las principales son: mainForm (que permite acceder a los objetos del formulario principal) y pioBoard (permite acceder al objeto para controlar el dispositivo STX80XX). Las variables, serán inicializadas cuando se instancie la clase UdpStreamRxCount desde el formulario principal a través de su constructor.

Otras variables son: samplesFile (archivo para almacenar las muestras), samplesCounter (mantiene el numero de muestras adquiridas), samplesMax (contiene el número máximo de muestras que deben adquirirse) y endOfSampling (valor boleano, que indica si el programa termino de almacenar las muestras).

Es importante notar, que los controles de la ventana del formulario principal (TextBox, ProgressBar, NumericUpDon, etc) , son accedidos desde esta clase a través del objeto “mainForm” pasado como referencia en el constructor de la clase.

La clase UdpStreamRxCount es la siguiente:

```
class UdpStreamRxVin : UdpStreamHandle
{
   /// <summary>
   /// Referencia al formulario principal.
   /// </summary>
   private Form1 mainForm = null;

   /// <summary>
   /// Referencia al objeto para controlar el dispositico STX80XX.
   /// </summary>
   private Stx8xxx pioBoard = null;

   /// <summary>
   /// Archivo para guardar las muestras recibidas.
   /// </summary>
   private StreamWriter samplesFile = null;

   /// <summary>
   /// Mantiene el numero de muestras recibidas.
   /// Cada muestra representa 1 byte.
   /// </summary>
   private UInt32 samplesCounter = 0;

   /// <summary>
   /// Numero maximo de muestras a adquirir.
   /// </summary>
   private UInt32 samplesMax;

   /// <summary>
   /// Indica si el programa termino de muestrear.
   /// </summary>
   private bool endOfSampling = false;
```

```
/// <summary>
/// Inicializa la clase.
/// </summary>
/// <param name="MainForm">Referencia al formulario principal Form1.</param>
/// <param name="PioBoard">Referencia al objeto para controlar el dispositivo STX80XX.</param>
public UdpStreamRxVin(Form1 MainForm, Stx8xxx PioBoard)
{
    // Guardamos la referencia al formulario principal.
    mainForm = MainForm;

    // Guardamos la referencia para controlar el dispositivo STX80XX.
    pioBoard = PioBoard;

    // Crear archivo para almacenar muestras.
    // El nombre lo obtenemos de la textBox1 del formulario.
    samplesFile = new StreamWriter(mainForm.textBox1.Text, false);

    // Obtener el numero maximo de muestras a adquirir.
    samplesMax = (UInt32) (mainForm.numericUpDown1.Value);

    // Inicializar ProgressBar.
    mainForm.progressBar1.Minimum = 0;
    mainForm.progressBar1.Maximum = (int) samplesMax;
    mainForm.progressBar1.Value = 0;

    // El programa no ha terminado la adquisicion.
    endOfSampling = false;

    // Establecer rango de 0-10V para canal VIN1 (llamar una sola vez para configurar).
    PioBoard.Cmd.Vin.SetRange(VinCh.Vin1, VinVoltRange.Unipolar_10V);
}

public override void RxHandle(UdpStreamPacket Packet, UdpStreamStat PacketStatus)
{
    // Obtenemos el identificador de paquete recibido.
    UdpStreamPacketIDs ID = (UdpStreamPacketIDs) Packet.ID;

    // Comprobar si el paquete UDP-STREAM fue recibido con exito.
    if (PacketStatus == UdpStreamStat.Success)
    {
        // Comprobar, si el paquete contiene muestras de entrada VIN1.
        if (ID == UdpStreamPacketIDs.VIN1)
        {
            // Guardamos el numero de muestras recibidas.
            samplesCounter += Packet.DataSize;

            // Variable para almacenar el votage de una muestra.
            float Voltage;
```

```
        // Escribir muestras del paquete en archivo.
        foreach (byte Sample in Packet.Data)
        {
            // Obtener voltage de la muestra (binario, resolucion 8 bits).
            Voltage = pioBoard.Cmd.Vin.BinToVoltage(VinCh.Vin1, Sample, 8);

            // Redondear voltage a 2 digitos decimales.
            Voltage = (float) Math.Round(Voltage, 2);

            // Escribir una muestra por linea.
            samplesFile.WriteLine(Voltage.ToString());
        }

        // Forzar escribir buffer del archivo en disco.
        samplesFile.Flush();

        // Detener muestreo ?.
        if (samplesCounter >= samplesMax)
        {
            // Desactivar sampler A.
            pioBoard.Cmd.Vin.SamplerAStop();

            // Cerrar archivo.
            samplesFile.Flush();
            samplesFile.Close();

            // Se ha terminado de adquirir datos.
            endOfSampling = true;

            // FIN: Mostrar mensaje.
            MessageBox.Show("Fin de Adquisicion!", "Completado...",
                                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            // Incrementar ProgressBar (asegurarse de no pasar el valor limite).
            mainForm.progressBar1.Value =
                        (int) (samplesCounter > samplesMax ? samplesMax : samplesCounter);
        }
    }
  }
}
```

```
    public override void RxStop()
    {
        // Desactivar sampler A.
        pioBoard.Cmd.Vin.SamplerAStop();

        // El programa no ha terminado de adquirir datos ?.
        if (endOfSampling == false)
        {
            // Cerrar archivo.
            samplesFile.Flush();
            samplesFile.Close();
        }
    }
}
```

Cuando instancie esta clase desde el formulario principal (Form1.cs), deberá pasar los objetos necesarios para acceder al formulario principal y al dispositivo STX80XX desde esta clase, de la siguiente forma:

```
// Instanciar Manejador de paquetes UDP-STREAM.
// Pasamos como referencia el formulario principal (this)
// y el objeto para controlar el dispositivo STX80XX en su argumentos.
UdpStreamHandler = new UdpStreamRxVin(this, PioBoard);
```

Y para que los controles del formulario (**TextBox**, **ProgressBar**, etc) sean visibles, deben ser públicos, entonces, en el entorno de desarrollo Microsoft Visual C#, vamos a la ventana de diseño del formulario principal y seleccionamos un **TextBox**. Luego vamos a la ventana de propiedades, y en el campo "**Modifiers**", seleccionamos "**Public**". Repetimos la operación con todos los controles del formulario. Este paso se explico con más detalle, en la sección de manejo para paquetes de entradas discretas.

10.4.3 Enviar Comandos al Dispositivo

A continuación, vamos a definir el constructor del formulario Form1.cs y los eventos click para el botón "Iniciar" y el botón "Detener" del formulario.

VARIABLES GLOBALES:

Objetos accesibles desde toda la clase Form1:

```
/// <summary>
/// Objeto para controlar el dispositivo STX80XX.
/// Power I/O Board.
/// </summary>
private Stx8xxx PioBoard;

/// <summary>
/// Objeto para controlar paquetes UDP-STREAM.
/// </summary>
private UdpStream UdpStreamControl = null;
```

```
/// <summary>
/// Manejador de paquetes UDP-STREAM.
/// </summary>
private UdpStreamRxVin UdpStreamHandler = null;
```

CONSTRUCTOR:

Aquí inicializamos variables y objetos del formulario principal:+

```
// Constructor del programa.
public Form1()
{
    // Requerido por Visual C#.
    InitializeComponent();

    // Instanciar objeto para controlar el dispositivo STX80XX.
    // Pasamos direccion IP, password y modelo de dispositivo utilizado.
    PioBoard = new Stx8xxx("192.168.1.81", 0, Stx8xxxId.STX8081);

    // Deshabilitar boton2 y habilitar boton1.
    button1.Enabled = true;
    button2.Enabled = false;
}
```

BOTON INICIAR (EVENTO CLICK):

Al hacer click sobre este botón, el programa debería escuchar paquetes UDP-STREAM y enviarle al dispositivo STX80XX la dirección IP de nuestra maquina, donde enviará los paquetes con las muestras adquiridas. Por defecto, se utilizó la dirección “192.168.1.11” como dirección de nuestra maquina, cámbiela a la dirección correcta de su máquina, y recompile el código.

```
// Iniciar Escucha de paquetes UDP-STREAM.
private void button1_Click(object sender, EventArgs e)
{
    // Instanciar Manejador de paquetes UDP-STREAM.
    // Pasamos como referencia el formulario principal (this)
    // y el objeto para controlar el dispositivo STX80XX en su argumentos.
    UdpStreamHandler = new UdpStreamRxVin(this, PioBoard);

    // Instanciar objeto para controlar paquetes UDP-STREAM.
    // Le pasamos como argumento el manejador de paquetes que
    // creamos para recibir los eventos "match" de los contadores.
    UdpStreamControl = new UdpStream(UdpStreamHandler);

    // Primero le especificamos al dispositivo STX80XX la direccion
    // IP de nuestra computadora, a la cual debe mandar los paquetes UDP-STREAM.
    PioBoard.Cmd.BoardConfig.SetUdpStreamHostIp(192, 168, 1, 11);
```

```
        // Configuramos la libreria para escuchar paquetes UDP-STREAM
        // en segundo plano.
        UdpStreamControl.StartListening();

        // Activar sampler A, muestrar entrada analogica VIN1 a 400 uS,
        // es decir a una tasa de muestreo de 2500 Hz.
        // Enviar las muestras obtenidas a traves de paquetes UDP-STREAM.
        PioBoard.Cmd.Vin.SamplerAStart(VinSamplerACh.Vin1, 400);

        // Deshabilitar boton1 y habilitar boton2.
        button1.Enabled = false;
        button2.Enabled = true;
    }
```

BOTON DETENER (EVENTO CLICK):

Cuando hacemos click en este botón, nuestra computadora deja de escuchar paquetes UDP-STREAM:

```
    // Detener Escucha de paquetes UDP-STREAM.
    private void button2_Click(object sender, EventArgs e)
    {
        // Al ejecutar este metodo, el metodo UdpStreamHandler.RxStop()
        // deberia ser llamado tambien, y el dispositivo dejaria de enviar
        // paquetes UDP-STREAM, debido a que se desactiva el sampler A.
        UdpStreamControl.StopListening();

        // Deshabilitar boton2 y habilitar boton1.
        button1.Enabled = true;
        button2.Enabled = false;
    }
```

10.4.4 Funcionamiento

Utilice el código de ejemplo provisto, compile el programa y pruébelo. No se comprobaran errores (por fines didácticos) por lo tanto verifique que el dispositivo conectado y funcionando correctamente, así como las direcciones IP empleadas sean las correctas.

Se recomienda inyectar una señal sinusoidal en la entrada VIN1, de un pico máximo de 6V y un pico mínimo de 1V. Una frecuencia de 100 Hz. Algunos modelos de dispositivo pueden requerir configuración extra del hardware (por ejemplo, en la STX8081 el J5 no debe estar colocado) o restringir el voltaje en las entradas (en el modelo STX8091 se recomienda utilizar la entrada VIN4 o reducir la tensión entre 1V y 4V). Consulte hojas de datos.

Inicie la escucha de paquetes UDP-STREAM, haciendo click en el botón “Iniciar”.



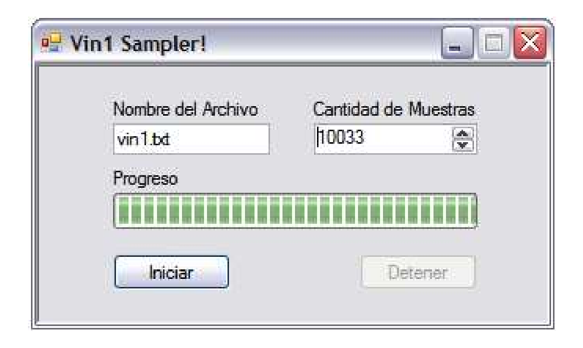
Comenzara la adquisición de datos. Notar como la barra de progreso se va completando:



Adquiridos los datos (10033 muestras), el programa indicara a través de una “MessageBox” lo siguiente:



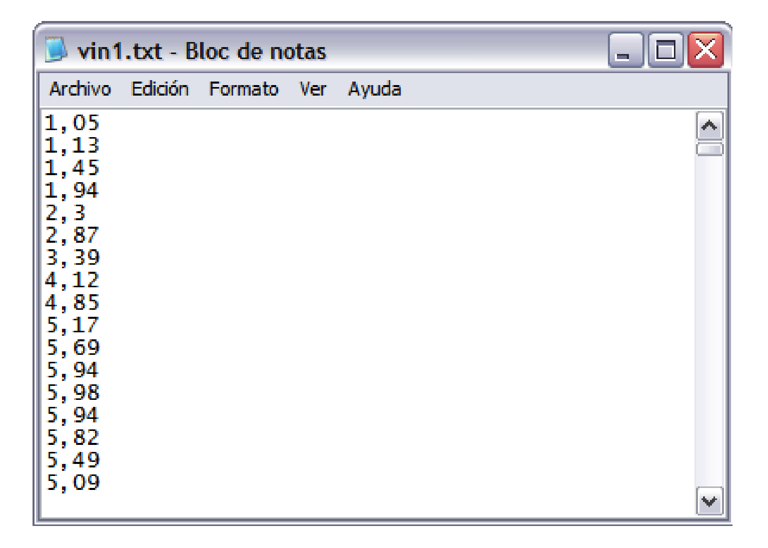
Click en el botón “Aceptar” de la “MessageBox” y luego detenga la escucha de paquetes UDP-STREAM, haciendo click en el botón “Detener”:



Cierre el programa.

Si las muestras fueron almacenadas correctamente en el archivo “vin1.txt”, es posible desde Microsoft Excel, importarlas y graficarlas.

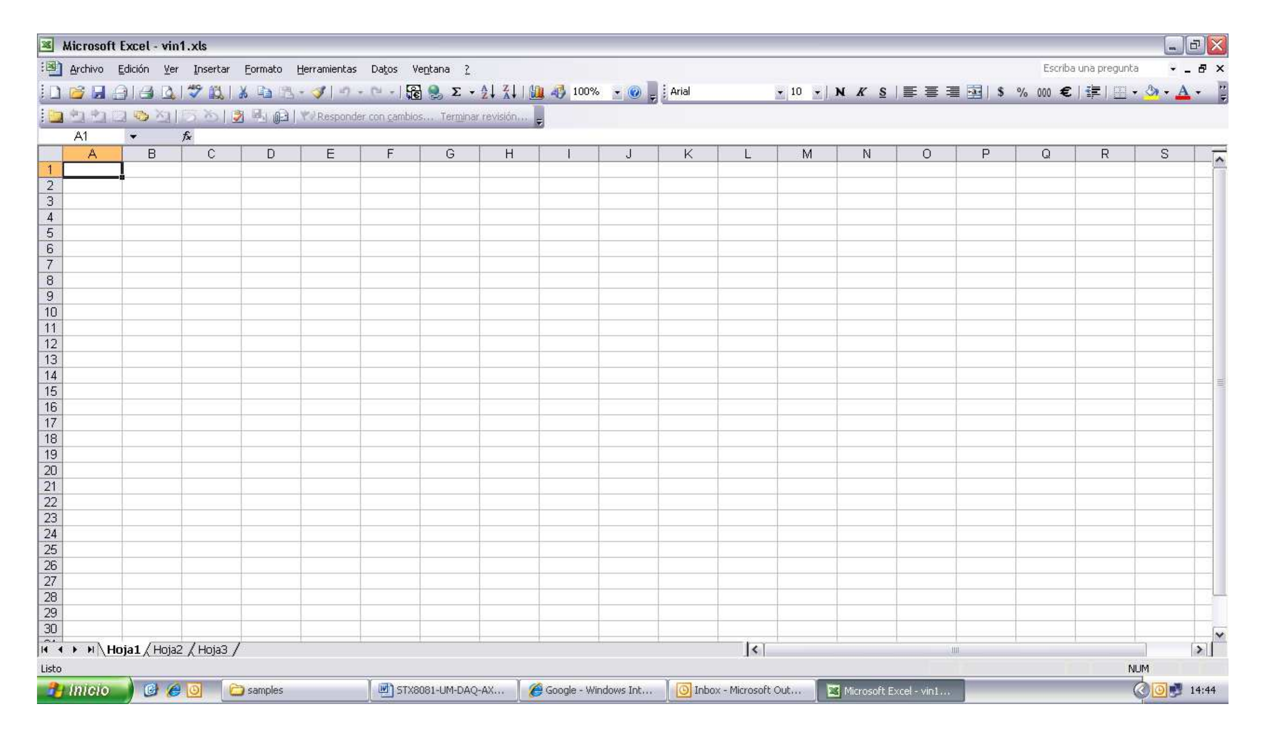
El archivo “vin1.txt”, puede ser abierto con el “Bloc de notas”, como se muestra a continuación:



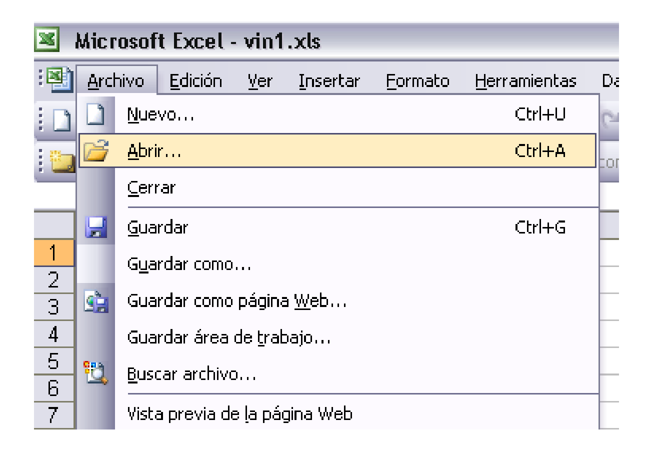
Notar, que cada muestra, fue almacenada en una línea separada, y contiene el voltaje de la entrada analógica.

10.4.5 Graficando las muestras en Microsoft Excel

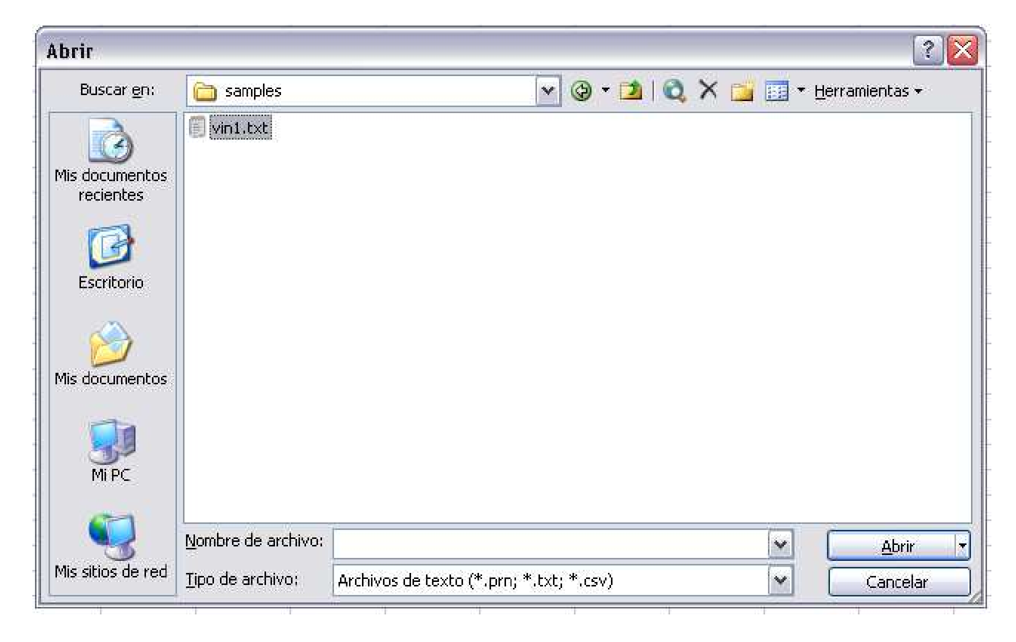
Abra el Excel:



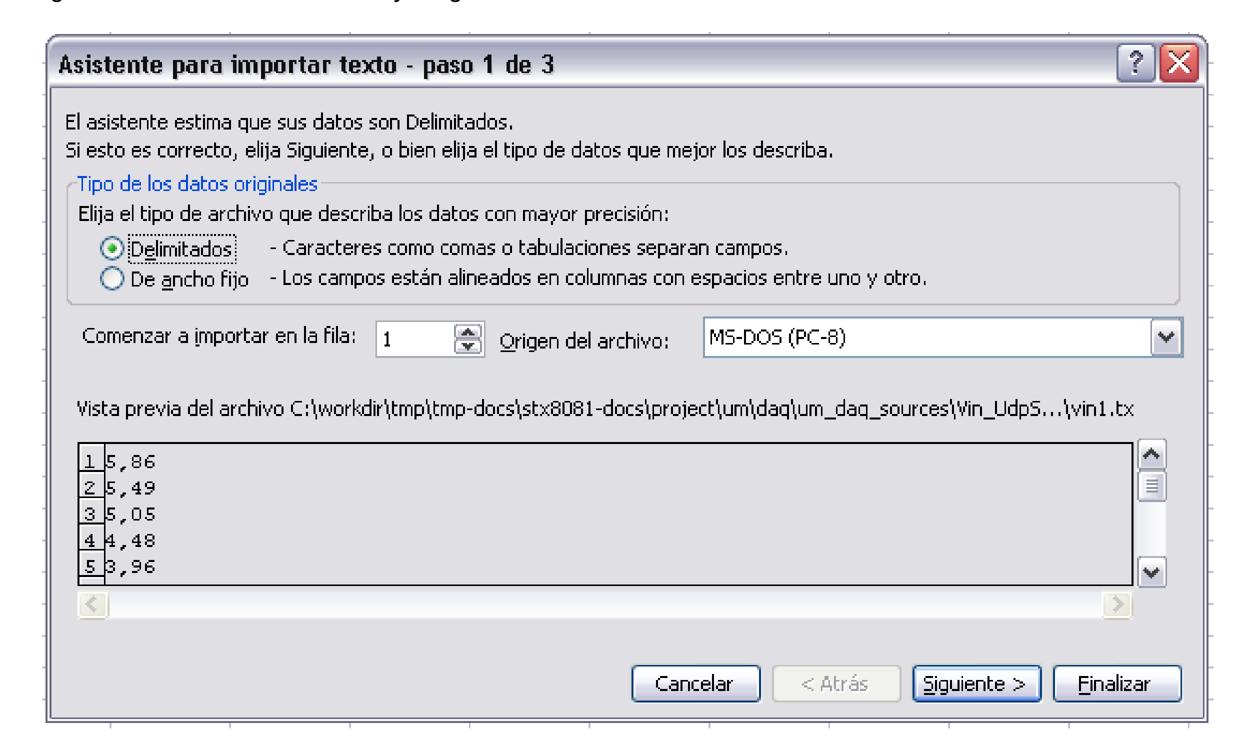
Seleccione “Archivo” y luego click en “Abrir…”:



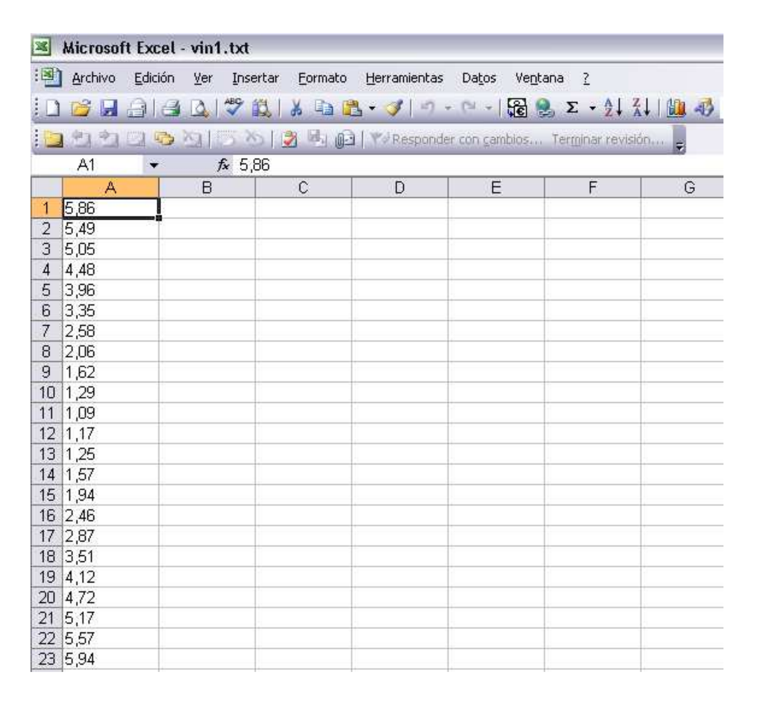
Cuando aparezca el cuadro de dialogo para abrir archivo, seleccione en “Tipo de Archivo”, el tipo “Archivos de texto (*.prn; *.txt; *.csv)” y busque el archivo vin1.txt:



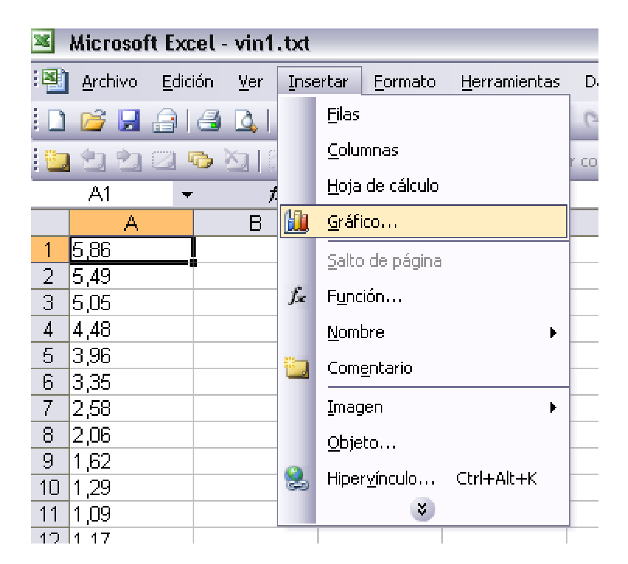
Finalmente, click en “Abrir”. Debería aparecer el siguiente cuadro. Seleccione en “Tipo de los datos originales” el ítem “Delimitados” y luego click en el botón “Finalizar”.



Si los datos se importan correctamente, cada muestra aparecerá en una fila por separado y en una misma columna, como se muestra a continuación:



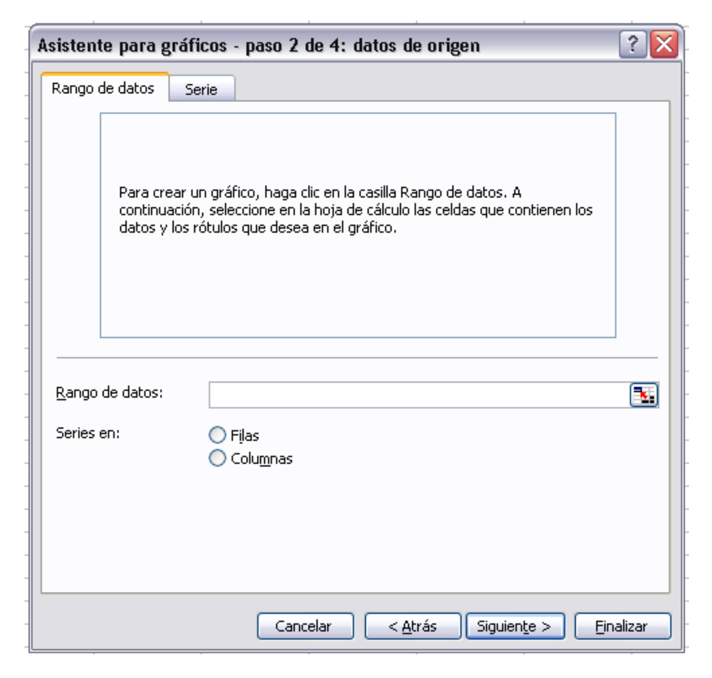
Para graficarlas, seleccione “Insertar” y luego click en “Grafico”:



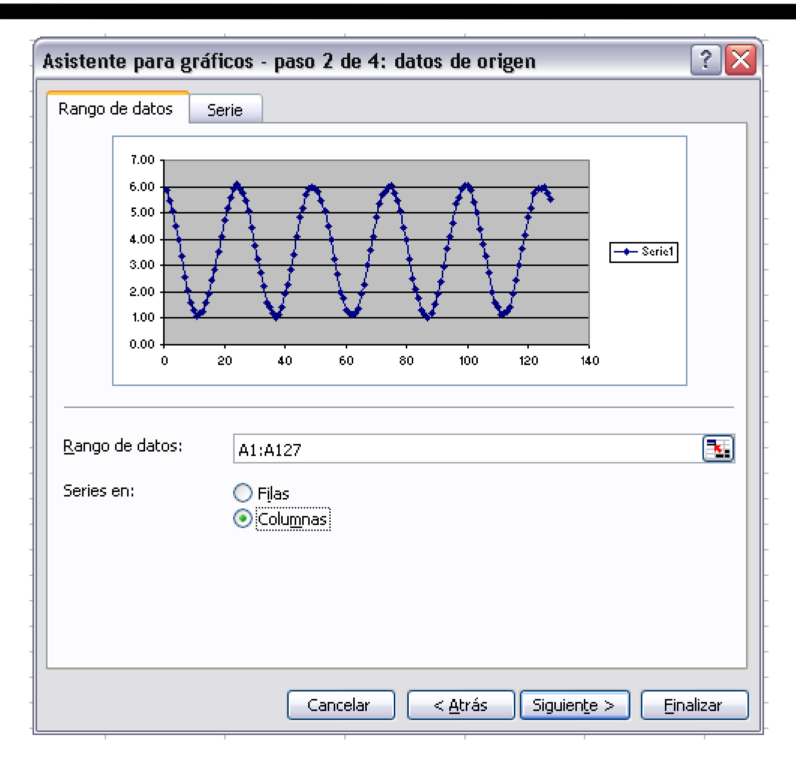
El siguiente cuadro de dialogo que se abrirá, seleccione “Tipo de Grafico” como “XY (Dispersión)” y el “Subtipo de grafico” como se muestra:



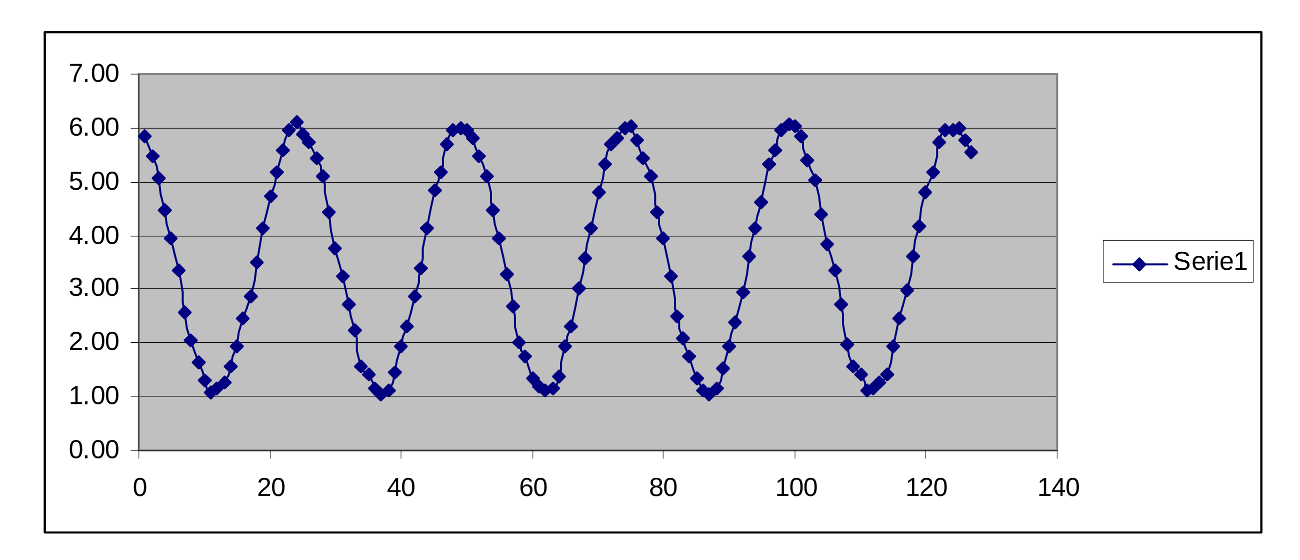
Luego, click en “Siguiente”. Otra pantalla aparecerá:



En rango de datos escribimos: “A1:A127”. Esto significa graficar las primeras 127 filas (es decir muestras). Luego seleccionamos el ítem “Columnas”:



Finalmente hacemos click en “Finalizar” y en el Excel quedara el siguiente grafico:



Como se aprecia en el grafico, una sinusoidal de pico máximo +6V y pico mínimo +1V es obtenida.

Nota: *De acuerdo a la configuración de su sistema, el punto decimal puede ser considerado como un punto “.” o una coma “;”. En ese caso, desde Excel deberá especificarlo de la forma adecuada con respecto a cómo fueron obtenidas las muestras. Un método simple, puede ser reemplazar todas las “,” por un punto “.”. Para ello click en menú “Edición” y luego en click en “Reemplazar”.*

10.4.6 Muestreando dos Entradas al Mismo Tiempo

El procedimiento es similar al empleado para una sola entrada, como fue explicado en secciones anteriores.

Si queremos muestrear el canal VIN1 y VIN2 al mismo tiempo, vamos al formulario principal y en el método para manejar el click del button1 activamos el sampler para muestrear ambos canales:

```
// Activar sampler A, muestrar entrada analogica VIN1 y VIN2 a 400 uS,
// es decir a una tasa de muestreo de 2500 Hz.
// Enviar las muestras obtenidas a traves de paquetes UDP-STREAM.
PioBoard.Cmd.Vin.Sampler12Start(VinSamplerACh.Vin1and2, 400);
```

En el método RxHandle() de la clase para manejar los paquetes recibidos, podemos almacenar las muestras de ambos canales, en dos archivos separados, como se muestra a continuación:

```
public override void RxHandle(UdpStreamPacket Packet, UdpStreamStat PacketStatus)
{
    // Obtenemos el identificador de paquete recibido.
    UdpStreamPacketIDs ID = (UdpStreamPacketIDs) Packet.ID;

    // Comprobar si el paquete UDP-STREAM fue recibido con exito.
    if (PacketStatus == UdpStreamStat.Success)
    {
        // Comprobar, si el paquete contiene muestras de entrada VIN1 y VIN2.
        if (ID == UdpStreamPacketIDs.VIN1AND2)
        {
            // Variables para almacenar el votage de una muestra.
            float Voltage;

            // Guardar las muestras del canal VIN1 (primera mitad del array).
            for (int i = 0; i < Packet.DataSize / 2; i++)
            {
                // Obtener voltage de la muestra (binario, resolucion 8 bits).
                Voltage = pioBoard.Cmd.Vin.BinToVoltage(VinCh.Vin1, Packet.Data[i], 8);

                // Redondear voltage a 2 digitos decimales.
                Voltage = (float) Math.Round(Voltage, 2);

                // Escribir una muestra por linea.
                samplesFile_Vin1.WriteLine(Voltage.ToString());
            }

            // Guardar las muestras del canal VIN2 (segunda mitad del array).
            for (int i = Packet.DataSize / 2; i < Packet.DataSize; i++)
            {
                // Obtener voltage de la muestra (binario, resolucion 8 bits).
                Voltage = pioBoard.Cmd.Vin.BinToVoltage(VinCh.Vin2, Packet.Data[i], 8);

                // Redondear voltage a 2 digitos decimales.
                Voltage = (float) Math.Round(Voltage, 2);
```

```
            // Escribir una muestra por linea.
            samplesFile_Vin2.WriteLine(Voltage.ToString());
        }

        // Forzar escribir buffer del archivo en disco.
        samplesFile_Vin1.Flush();
        samplesFile_Vin2.Flush();
    }
  }
}
```

10.4.7 Graficando Muestras con Librería

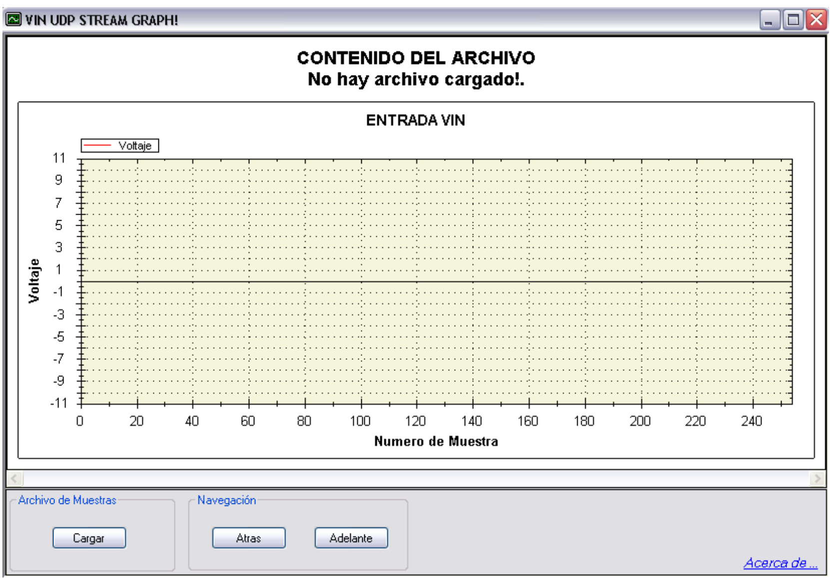
Se adjunta un proyecto llamado “Vin_UdpStream_Graph”, en el cual es posible graficar las muestras obtenidas con el programa “Vin_UdpStream”, desarrollado con anterioridad.

Anteriormente, las muestras fueron graficadas utilizando Microsoft Excel. Si bien es una alternativa útil para generar informes, muchas veces es necesario graficar las muestras en nuestro programa C#.

Para graficar, utilizamos la librería **ZedGraph**, de uso libre y gratuito. Puede ser descargada desde http://zedgraph.org, junto con documentación y ejemplos de uso. En www.google.com puede buscar más ayuda aun. El kit de desarrollo de software (STX80XX-SDK) incluye también esta librería, para fácil acceso, en el directorio “**visual_cs\libs\zedgraph**”. En el directorio “**visual_cs\libs\zedgraph_doc**” se encuentra el manual de la librería.

Su uso es simple, sin embargo, no explicaremos los detalles de la librería, ya que excede este documento.

El programa carga un archivo de muestras obtenidas con el programa “Vin_UdpStream” y las grafica en un panel. El usuario puede desplazarse en las muestras, efectuar Zoom, etc.



Pantalla Principal

Para graficar muestras, primero cargue el archivo donde se encuentran las muestras adquiridas, por ejemplo en “vin1.txt”. Para ello, haga click en el botón “Cargar” del programa y navegue los directorios hasta encontrar el archivo:

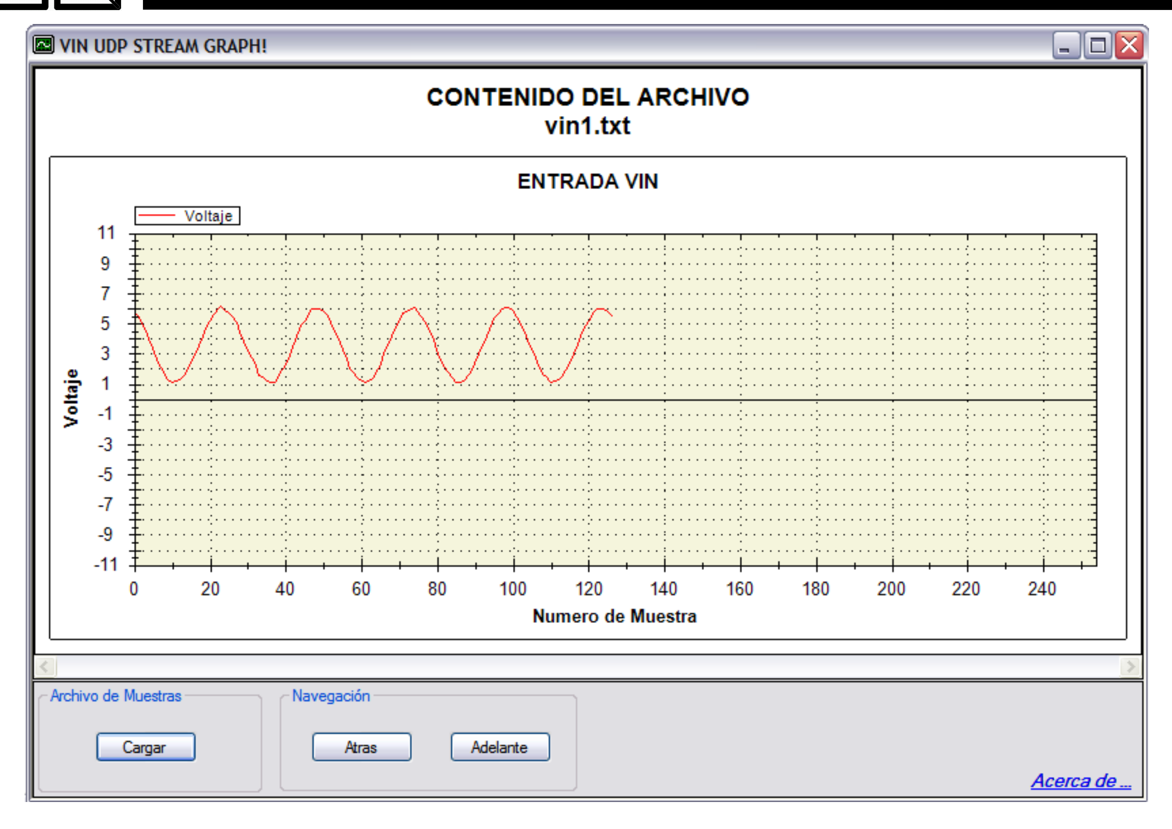
1. Cargar Archivo

2. Luego Seleccione Archivo

3. Finalmente Abra el Archivo

A continuación, se muestra como el programa representa la grafica:

© 2015 Slicetex Electronics

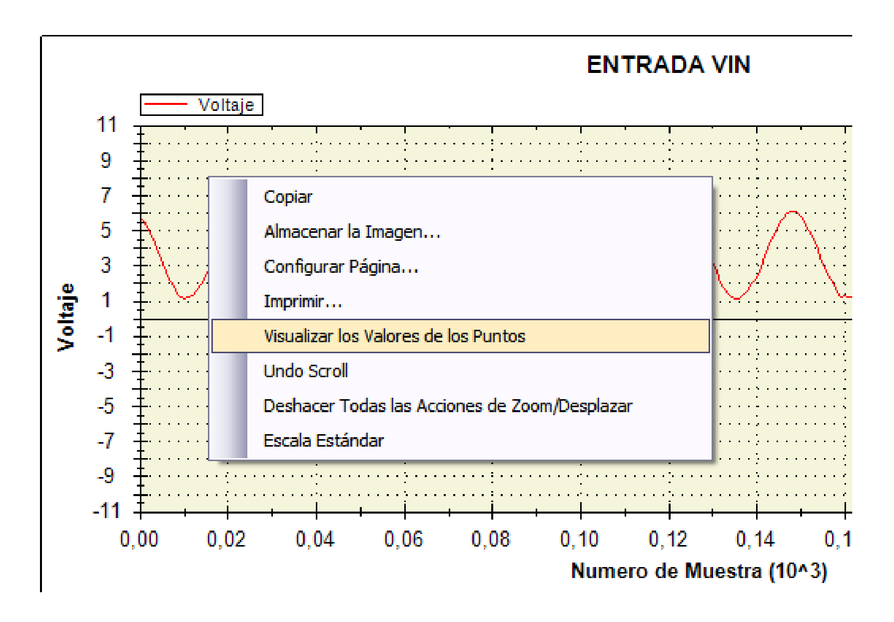


Si hacemos click en “Adelante” cargamos las próximas 127 muestras desde el archivo cargado. Un cuadro de texto nos avisara al llegar al final del archivo.

Si hacemos click en “Atrás”, nos desplazamos hacia las 127 más antiguas del grafico. Un cuadro de texto nos avisara al llegar al inicio del archivo.

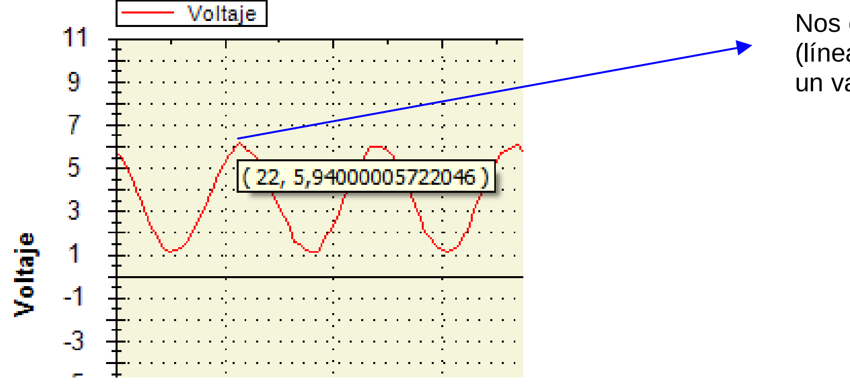
Nota: *Este programa interpreta que las muestras utilizan el punto decimal definido por el sistema. Si el punto decimal (coma o punto) de su sistema es distinto al punto decimal utilizado al programa que genero el archivo “vin1.txt”, la grafica será errónea. En ese caso o tendrá que cambiar las comas por punto o viceversa en el archivo “vin1.txt”.*

Al hacer un click derecho sobre el grafico, obtenemos una lista de opciones útiles:



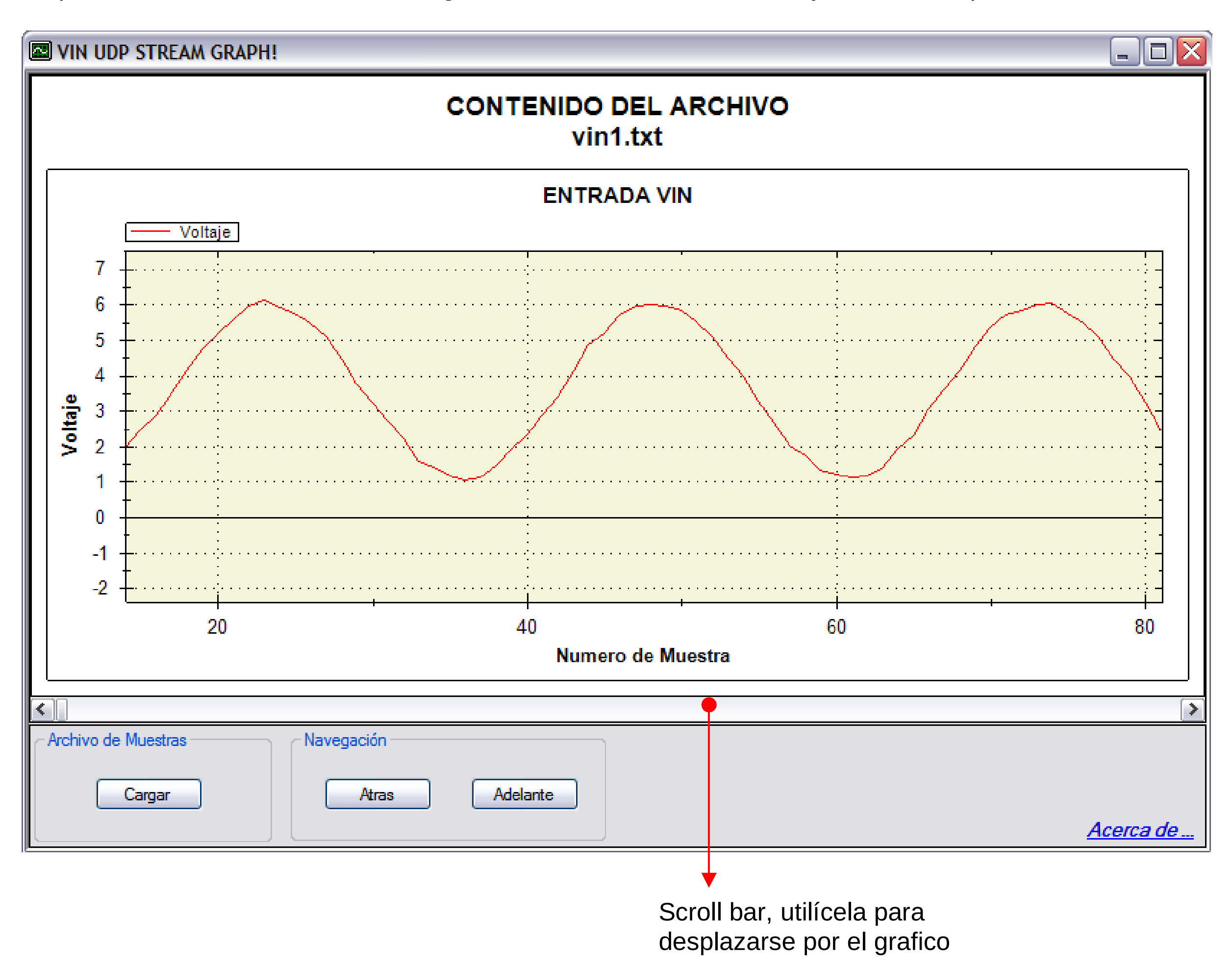
Podemos copiar la imagen, Almacenar la imagen en disco duro, imprimirla, Visualizar los Valores de los Puntos, etc.

Elegimos “Visualizar los Valores de los Puntos” para ver el valor de la muestra, al pasar el Mouse sobre el grafico:



Nos dice que: La muestra “22” (línea 22 del archivo vin1.txt,) tiene un valor de 5.94Volts.

Es posible efectuar un Zoom a la imagen, haciendo clic con el Mouse y mientras se presiona, arrastrarlo:

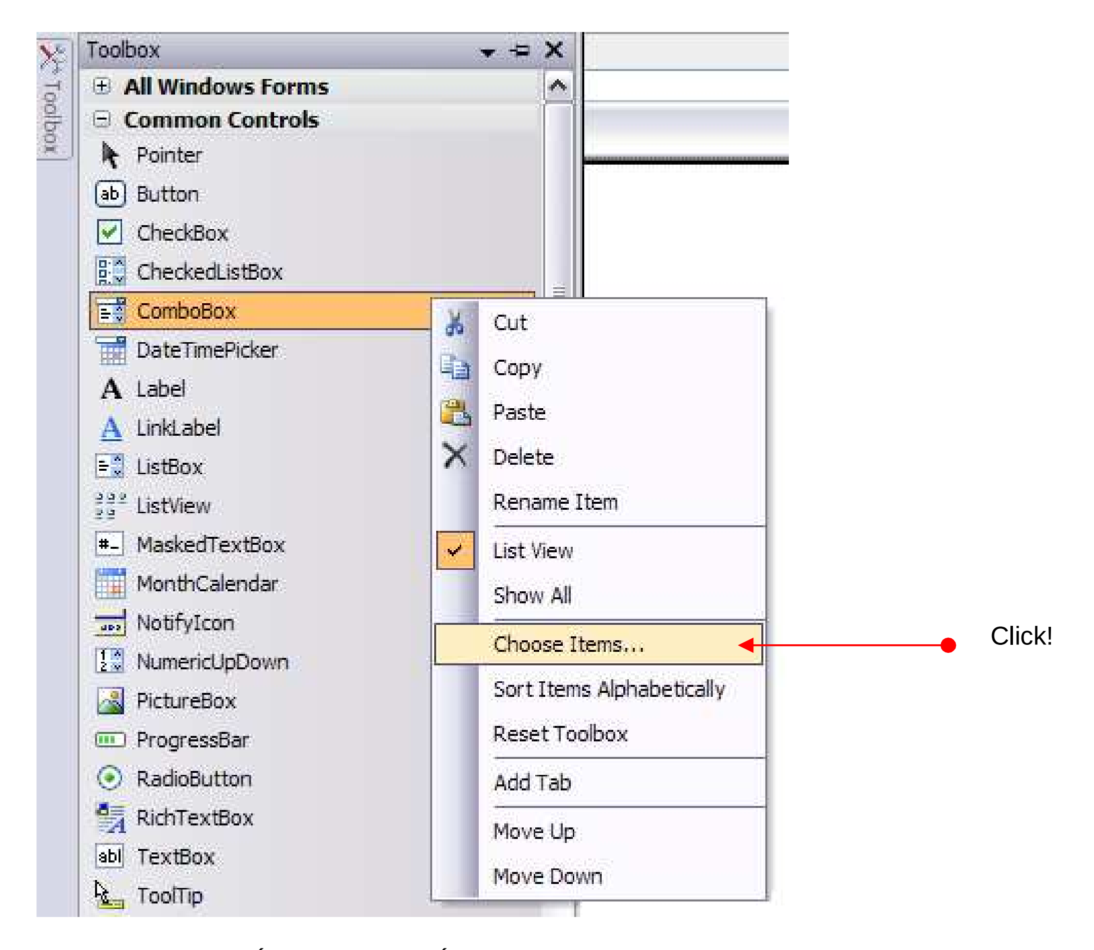


Luego del zoom, notamos que la señal tiene un pico máximo de +6V y un pico mínimo de +1V, tal como la señal de entrada.

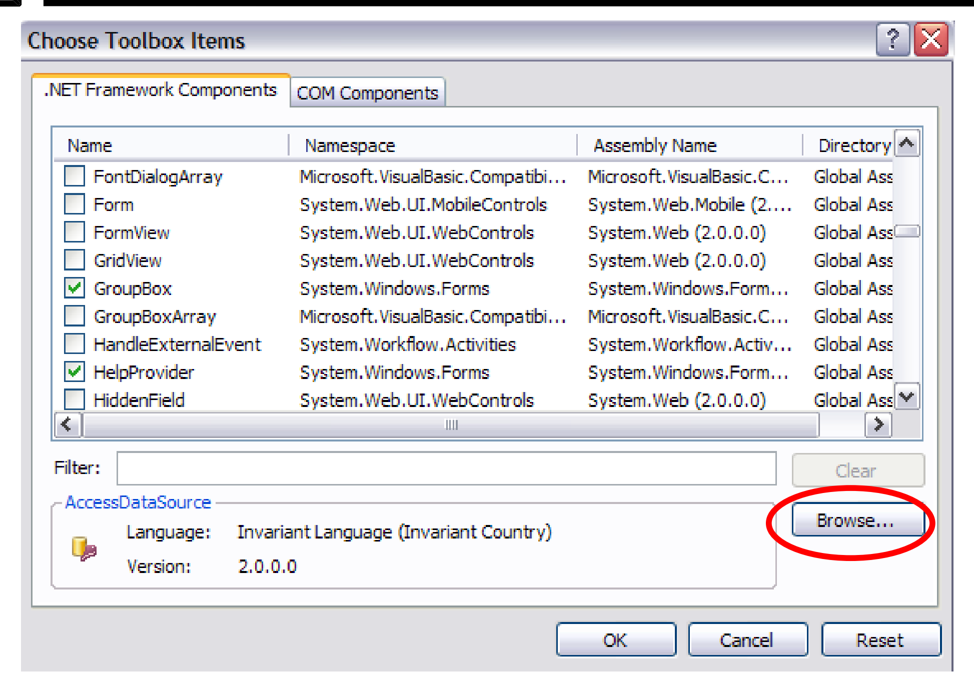
10.4.8 Añadiendo ZedGragh a su Proyecto

Para utilizar la librería ZedGraph en su proyecto debe agregar como referencia el archivo “ZedGraph.dll” y “ZedGraph.Web.dll” en su proyecto, de la misma forma en que agrego la librería “stx8xxx.dll” para utilizar el dispositivo. El procedimiento se explicó al comienzo de este documento. Los archivos “ZedGraph.dll” y “ZedGraph.Web.dll” se pueden localizar en el directorio “**visual_cs\libs\zedgraph**” del STX80XX-SDK.

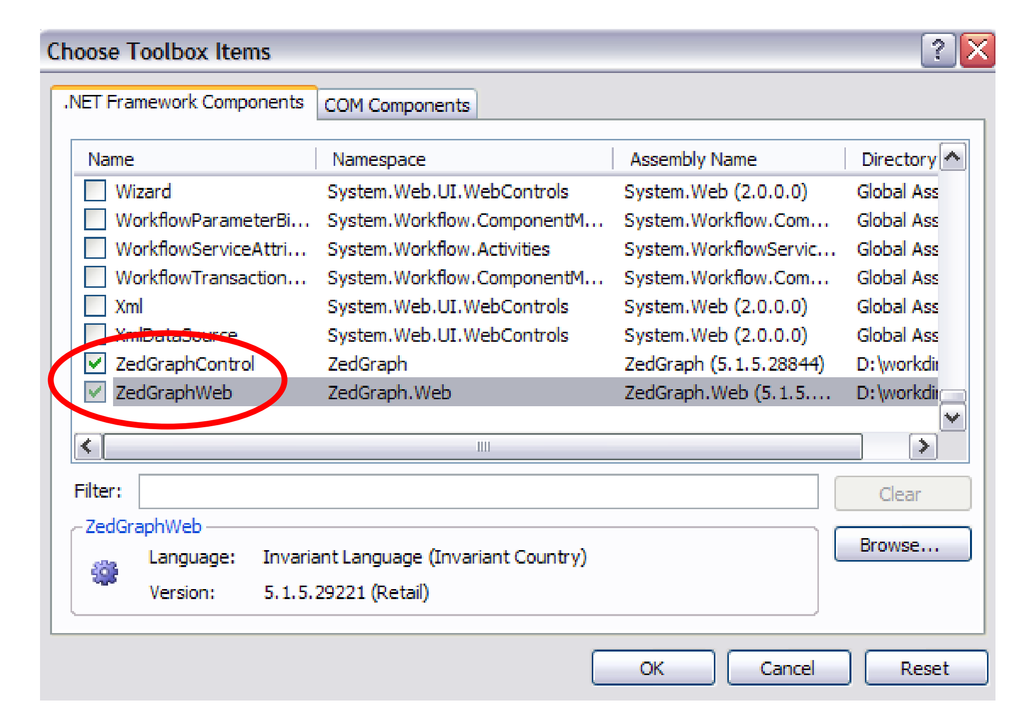
Finalmente, los controles gráficos se agregan haciendo clic derecho sobre la caja de herramientas desde un formulario:



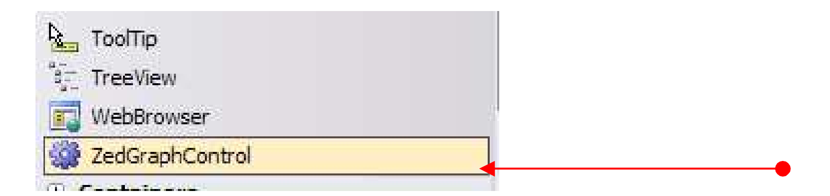
Seleccionar “Cose Ítems” o “Elegir Ítems”, luego se abrirá la siguiente ventana:



Seleccione “Browse…” o “Buscar…”, y busque los archivos “ZedGraph.dll” y “ZedGraph.Web.dll”, para agregarlos.



Tilde los ítems “ZedGraphControl” y “ZedGraphWeb” de la nueva ventana y luego haga clic en “OK”.
Los controles estarán listos para utilizarse como se muestran a continuación:



En la caja de herramientas, ahora aparece el control "ZedGraphControl" que puede utilizar en sus formularios, para graficar utilizando la librería ZedGraph.

11 Abreviaciones y Términos Empleados

- **PLC**: Programable Logic Controller (Controlador Lógico Programable).
- **DAQ**: Data Aquisition (Adquisición de Datos).
- **Modo** PLC: Permite programar el dispositivo STX80XX mediante lenguaje Pawn o Ladder y ejecutarlos autónomamente para realizar algún tipo de control.
- **Modo DAQ**: Permite controlar el dispositivo STX80XX a través de una computadora conectada a la interfaz Ethernet, ya sea para adquirir datos o controlar las salidas del dispositivo.
- **UDP**: User Datagram Protocol. Protocolo orientado a la transmisión/recepción de datos. En el dispositivo STX80XX se usa para intercambiar datos mediante la interfaz Ethernet.
- **Bootloader**: Programa que corre en el dispositivo y permite actualizar el firmware.
- **Firmware**: Programa embebido el dispositivo y que contiene la lógica de funcionamiento.
- **IP**: Dirección Internet, conformada por cuatro octetos, por ejemplo 192.168.1.81.
- **UDP-STREAM**: Paquete transmitido desde el dispositivo STX80XX de forma asíncrona, que contiene información sobre algún evento.
- **Sampler**: Modulo lógico del dispositivo STX80XX que permite muestrear una entrada analógica a una tasa fija.

12 Historial de Revisiones

Tabla: Historia de Revisiones del Documento

Revisión	Cambios	Descripción	Estado
06 08/MAR/2015	1	1. Documentación adaptada a línea STX80XX. 2. Nuevo constructor de librería. 3. Nuevas funciones DOUT, VIN, VOUT, etc.	Preliminar
05 16/OCT/2014	1	1. Descripción para método FeedWatchdog() que permite activar y alimentar el watchdog del dispositivo, ver sección en pag. 99.	Preliminar
04 15/SEP/2012	1	1. Correcciones mínimas y adaptación a nuevo entorno StxLadder.	Preliminar
03 10/JUL/2012	1	1. Se agregan métodos para filtrar digitalmente las entradas analógicas, ver pag. 76.	Preliminar
02 31/AGO/2010	1	1. Correcciones varias.	Preliminar
01 04/JUN/2010	1	4. Versión preliminar liberada.	Preliminar

© 2015 Slicetex Electronics

13 Referencias

Ninguna.

14 Información Legal

14.1 Aviso de exención de responsabilidad

General: La información de este documento se da en buena fe, y se considera precisa y confiable. Sin embargo, Slicetex Electronics no da ninguna representación ni garantía, expresa o implícita, en cuanto a la exactitud o integridad de dicha información y no tendrá ninguna responsabilidad por las consecuencias del uso de la información proporcionada.

El derecho a realizar cambios: Slicetex Electronics se reserva el derecho de hacer cambios en la información publicada en este documento, incluyendo, especificaciones y descripciones de los productos, en cualquier momento y sin previo aviso. Este documento anula y sustituye toda la información proporcionada con anterioridad a la publicación de este documento.

Idoneidad para el uso: Los productos de Slicetex Electronics no están diseñados, autorizados o garantizados para su uso en aeronaves, área médica, entorno militar, entorno espacial o equipo de apoyo de vida, ni en las aplicaciones donde el fallo o mal funcionamiento de un producto de Slicetex Electronics pueda resultar en lesiones personales, muerte o daños materiales o ambientales graves. Slicetex Electronics no acepta ninguna responsabilidad por la inclusión y / o el uso de productos de Slicetex Electronics en tales equipos o aplicaciones (mencionados con anterioridad) y por lo tanto dicha inclusión y / o uso es exclusiva responsabilidad del cliente.

Aplicaciones: Las aplicaciones que aquí se describen o por cualquiera de estos productos son para fines ilustrativos. Slicetex Electronics no ofrece representación o garantía de que dichas aplicaciones serán adecuadas para el uso especificado, sin haber realizado más pruebas o modificaciones.

Los valores límites o máximos: Estrés por encima de uno o más valores límites (como se define en los valores absolutos máximos de la norma IEC 60134) puede causar daño permanente al dispositivo. Los valores límite son calificaciones de estrés solamente y el funcionamiento del dispositivo en esta o cualquier otra condición por encima de las indicadas en las secciones de Características de este documento, no está previsto ni garantizado. La exposición a los valores limitantes por períodos prolongados puede afectar la fiabilidad del dispositivo.

Documento: Prohibida la modificación de este documento en cualquier medio electrónico o impreso, sin autorización previa de Slicetex Electronics por escrito.

15 Información de Contacto

Para mayor información, visítenos en www.slicetex.com

Para información general, consultas y ventas, envíe un mail a: info@slicetex.com

Para soporte técnico ingrese a nuestro foro en: www.slicetex.com/foro

Ing. Boris Estudiez

Slicetex Electronics
Córdoba, Argentina

© Slicetex Electronics, todos los derechos reservados.

16 Contenido